Proceedings of the Seminar

Artificial Intelligence, Natural Language Processing and Information Retrieval

University of Colorado, Colorado Springs

August 7, 2010

Editors: Jugal Kalita and Terrance Boult

Funded by

National Science Foundation

Reducing Training Time for Linear SVMs

Nick Arnosti, Williams College

August 6, 2010

Abstract—**Support Vector Machines (SVMs) have been shown to achieve high performance on classification-related tasks, and are generally portable across domains. Because training time for SVMs typically scales poorly as the size of the training set increases, a great deal of effort has been dedicated to reducing the time required to learn a classification model. Recent work presented by Joachims in [1], and improved by Franc and Sonnenburg in [2] and [3], makes use of a reformulation of the SVM primal problem and cutting-plane approximation algorithms to train linear SVMs in time linear with respect to the size of the training set. In this paper, we observe an inefficiency in the algorithms presented in the above works, and introduce a modification which reduces training time for linear SVMs by up to 40% when compared to the methods presented in [2] and [3]. Additionally, we analyze the effect that runtime parameters and data set attributes have on the performance of our algorithms.**

I. Introduction

When using an SVM to classify objects, it must be provided with training data. This consists of a set of examples of the form $(\mathbf{x}_i, y_i)$, where each $\mathbf{x}_i$ is a vector of feature values (independent variables) taken from the space $\mathcal{X}$, and y_i is a class label (the dependent variable) taken from the set $\mathcal{Y}$. The idealized goal, then, is to come up with a function $g : \mathcal{X} \to \mathcal{Y}$ such that for each i, $g(\mathbf{x}_i) = y_i$. Throughout this paper, n is the number of training examples, s the number of features for each example, $\mathcal{X} = \mathbb{R}^s$, and k is the number of distinct classes.

For binary classification, $\mathcal{Y} = \{-1, +1\}$, and the SVM is biased to look for solutions of the form $g(\mathbf{x}_i) = sgn(\mathbf{w}^T\mathbf{x}_i + b)$ for fixed $\mathbf{w} \in \mathbb{R}^f, b \in \mathbb{R}$ (from [4]). For linearly separable data sets, the goal is to find the choice of $\mathbf{w}$ which maximizes the margin between the two classes. This is equivalent to minimizing $||\mathbf{w}||$ subject to the constraints $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1\ \forall i$.

In general, the data is not linearly separable, so it is necessary to introduce error terms $\xi_i \geq 0$ for each training example. The goal becomes to maximize the margin between classes while also minimizing error on the training set. Using an L1-loss function, this boils down to the following optimization problem (from [5]):

$$\begin{array}{ll} \text{minimize:} & \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum_{i=1}^n \xi_i \\ \text{subject to:} & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i\ \forall i, \end{array} \tag{1}$$

where $C \in \mathbb{R}$ is known as the regularization constant and determines the relative importance of the two objectives.

One unfortunate result of this formulation is that because we have with n error terms ξ_i, optimizing them takes $\mathcal{O}(n^2)$ time, making SVM training a very slow process on large data sets. As a result, most well-known SVM solvers, such as LIBSVM (http://www.csie.ntu.edu.tw/ cjlin/libsvm/), scale poorly as the size of the training set increases.

Many multi-class solutions simply break the problem down into a series of binary classification problems, and then create a binary classifier to solve each of these. The simplest and most common approach is referred to as *one-against-all*, which trains one binary SVM, $\mathbf{w}_y$, for each class $y \in \mathcal{Y}$, using as input labels $+1$ if $y_i = y$ and -1 otherwise. The decision function in this case is $y_{predicted} = \text{argmax}_{y\in\mathcal{Y}}(\mathbf{w}_y^T\mathbf{x}_i + b_y)$. Other common techniques are known as *one-against-one*, *half-against-half* and *error-correcting output coding*, described in [6], [7] and [8], respectively. It is noted in [9] that for many data sets, these approaches do not produce results that are as good as coming up with a true multi-class solution.

In the multi-class approach presented in [10], instead of one slack variable for each point in the training set, there are k of them. Making the simplifying (and easily reversible) assumption that each $b_y = 0$, let ξ_{ij} denote the error of point i with respect to class j, and $\mathbf{w}_j$ denote the linear classifier corresponding to class j. Then the optimization problem to be solved is of the form:

$$\begin{array}{ll} \text{minimize:} & \frac{1}{2}\sum_{j=1}^k \mathbf{w}_j^T\mathbf{w}_j + C\sum_{i=1}^n\sum_{j=1}^k \xi_{ij} \\ \text{subject to:} & \mathbf{w}_j^T\mathbf{x}_i \geq 1 - \xi_{ij}\ \forall i, j. \end{array} \tag{2}$$

Unfortunately, due to the large number of slack variables, solving this optimization problem exactly is very computationally intensive. In [9], the number of slack variables is reduced to n by defining $\xi_i = \max_j\{\xi_{ij}\}$. This simplifies the form of the optimization problem above, but optimizing the $n \times k$ matrix $\mathbf{w}$ remains a time-intensive task as the number of classes and training examples grows. Recent research ([1],[2],[11]) has focused on finding approximate solutions, and has resulted in a significant decrease in SVM training time. Section II discusses the algorithms presented in some of this recent work. Section III makes note of one way in which the algorithms presented in [1] and [2] are sub-optimal and introduces two new algorithms which make use of this observation. We tested our algorithms on several data sets, and the results of these tests are presented and analyzed in Section IV, with additional data provided in the Appendix. In the process of conducting tests, we stumbled across an unexpected benefit to our algorithms, which is presented in Section V. We close with a brief conclusion and discussion of possibilities for future work.

II. Previous Work

In [1], Joachims presents what he calls the "structural formulation" of the primal (1), which reduces the number of slack variables from n to 1. For mathematical simplification, Joachims makes the assumption that $b = 0$, which can easily be reversed by adding a feature of weight 1 to each vector $\mathbf{x}_i$. This formulation can be written as:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi \\ \text{subject to} & \frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i) \leq \xi \end{array} \tag{3}$$

Joachims shows that (1) and (3) are mathematically equivalent after setting $\xi = \frac{1}{n}\sum_{i=1}^{n}\xi_i$, meaning that a vector $\mathbf{w}^*$ is a solution to one if and only if it is a solution to the other. This structural formulation made it possible to come up with very good approximations of the optimal objective value in time linear with respect to the size of the training set - a significant improvement upon past performance.

In the years since Joachims introduced this structural formulation, several others have looked to extend and improve upon his work. One notably successful implementation is the Optimized Cutting Plane Algorithm for Support Vector Machines, or OCA. This paper will adopt the notation used in [2], where the objective function from (3) is given the name $F(\mathbf{w})$, and the expression on the left hand side of the constraint, which represents the risk using an L1-loss function, is dubbed $R(\mathbf{w})$. Rather than solving the problem directly, Joachims' cutting-plane algorithm (henceforth referred to as CPA) creates ever-better approximations $R_t(\mathbf{w})$ to $R(\mathbf{w})$, where t indicates the number of iterations.

In [1], Joachims conceives of the constraint from (3) as 2^n distinct constraints. Each iteration modifies $R_t(\mathbf{w})$ by taking into account the "most violated constraint." In the language of [2], this corresponds to taking a subgradient of $R(\mathbf{w})$. Given a choice of $\mathbf{w}'$, the subgradient of interest $\mathbf{a}'$ can be computed as follows:

$$\mathbf{a}' = -\frac{1}{n}\sum_{i=1}^{n}\pi_i y_i \mathbf{x}_i \quad \text{where } \pi_i = \begin{cases} 1 & \text{if } y_i\langle\mathbf{w}', \mathbf{x}_i\rangle < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Because R is a convex function, for any point $\mathbf{w}'$ and the corresponding subgradient $\mathbf{a}'$, the linear approximation $R(\mathbf{w}') + \langle\mathbf{a}', \mathbf{w} - \mathbf{w}'\rangle$ provides a lower bound for $R(\mathbf{w})$. For notational simplicity, the authors of [2] define $b' = R(\mathbf{w}') - \langle\mathbf{a}', \mathbf{w}'\rangle$, so that $R(\mathbf{w}) \geq \langle\mathbf{a}', \mathbf{w}\rangle + b'$. Given a collection of cutting planes defined by pairs $(\mathbf{a}_i, b_i)$, $R_t(\mathbf{w})$ is their pointwise maximum (subjected to a non-negativity constraint): $R_t(\mathbf{w}) = \max(0, \max_{1\leq i\leq t}\{\langle\mathbf{a}_i, \mathbf{w}\rangle + b_i\})$.

One iteration of Joachims' CPA computes $\mathbf{w}_{t+1}$ by solving the simplified optimization problem generated by substituting $R_t(\mathbf{w})$ for $R(\mathbf{w})$. Then $\mathbf{a}_{t+1}$ and b_{t+1} are computed by the rules above. The algorithm terminates when $1 - \frac{F_t(\mathbf{w}_t)}{F(\mathbf{w}_t)} \leq \epsilon$, a constant provided as a parameter to the solver. This guarantees that $F(\mathbf{w}_t)(1-\epsilon) \leq F(\mathbf{w}^*)$, where $F(\mathbf{w}^*)$ is the solution to (1).

There are several advantages to this technique. Computing $\mathbf{a}_i$ and b_i takes $\mathcal{O}(ns)$ time. In [1], Joachims proves that the number of iterations required to reach ϵ-precision depends only on ϵ (he bounds the growth at a rate of $\mathcal{O}(C/\epsilon)$) and not on the number of training examples n. The time required to solve the reduced optimization problem grows superlinearly with the number of iterations, but as the number of constraints is bounded by a constant independent of n and s, CPA runs in $\mathcal{O}(ns)$ time. Further details and analysis of CPA can be found in [12].

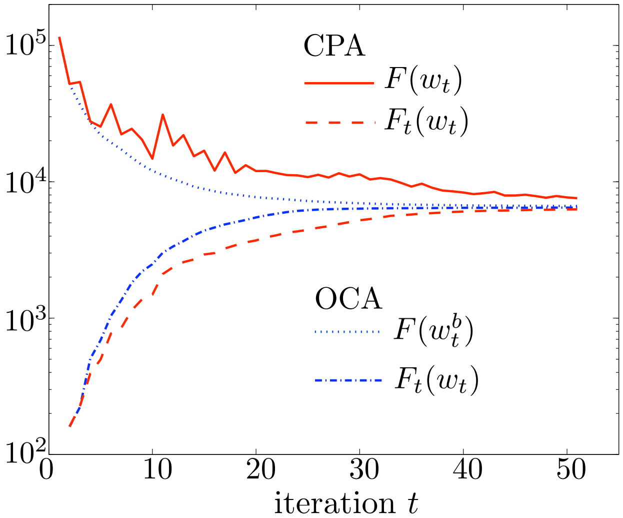


Fig. 1. The effect of the OCA changes, with objective value on the y-axis. Note that the CPA overestimate $F(\mathbf{w}_t)$ fluctuates significantly, while the OCA overestimate $F(\mathbf{w}_t^b)$ decreases monotonically, causing faster convergence.

In [2], the authors note several ways in which CPA is sub-optimal. Most notably, while the under-estimate $F_t(\mathbf{w}_t)$ increases monotonically with respect to t, the over-estimate $F(\mathbf{w}_t)$ fluctuates significantly, which can lead to slow rates of convergence. To solve this problem, a new value $\mathbf{w}_t^b$ is defined by $\mathbf{w}_t^b = \text{argmin}_{\mu\geq 0}F(\mu\mathbf{w}_t + (1-\mu)\mathbf{w}_{t-1}^b)$. Note that for $\mu = 0$, the argument to F is $\mathbf{w}_{t-1}^b$, so we have that $F(\mathbf{w}_t^b) \leq F(\mathbf{w}_{t-1}^b)$, and the over-estimate $F(\mathbf{w}_t^b)$ decreases monotonically. A visualization of the difference is taken from [2] and presented in Figure 1. While this modification does not change the growth rate, $\mathcal{O}(C/\epsilon)$, for the number of iterations, in practice OCA requires many fewer iterations to converge than CPA does. As a result, OCA training time is significantly less than that of CPA, often by an order of magnitude or more [2].

For notational convenience, define $f(\mu) = F(\mu\mathbf{w}_t + (1-\mu)\mathbf{w}_{t-1}^b)$. The process of finding the value of μ which minimizes f shall be referred as the "line search." It is shown in [2] that $\partial f(\mu)$ is a piecewise linear function with discontinuities at n values of μ. Solving $\partial f(\mu) = 0$ requires finding the points of discontinuity (each one can be computed in $\mathcal{O}(f)$ time), sorting them (which requires $\mathcal{O}(n\log n)$ time), and then evaluating the function at some of these points to determine where $\partial f(\mu)$ crosses the x-axis.

In [3], the authors present their multi-class implementation of OCA. In this formulation $\mathbf{w} \in \mathbb{R}^d$ (rather than $\mathbb{R}^f$) for some choice of d, and given a function $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$, define the decision function by $h_{\mathbf{w}}(\mathbf{x}) = \text{argmax}_{y\in\mathcal{Y}}\langle\mathbf{w}, \Psi(\mathbf{x}, y)\rangle$. Again, the objective function $F(\mathbf{w})$ is $\frac{1}{2}\mathbf{w}^T\mathbf{w} + CR(\mathbf{w})$, where risk $R(\mathbf{w})$ is defined by:

$$\frac{1}{n}\sum_{i=1}^{n}\max_{y\in\mathcal{Y}}(\delta(y, y_i) + \langle\Psi(\mathbf{x}_i, y) - \Psi(\mathbf{x}_i, y_i), \mathbf{w}\rangle) \tag{5}$$

tonically increases w.r.t. the number of iterations t. However, there is no such g

master problem objective $F(\mathbf{w}_t)$. Even though it will ultimately converge arbitrar

minimum $F(\mathbf{w}^*)$, its value can heavily fluctuate between iterations (Figure 1). The

Figure 1: Convergence behavior of the standard CPA vs. the proposed O

fluctuations is that at each iteration t, CPA selects the cutting plane that perfectly a master objective F *at the current solution* $\mathbf{w}_t$. However, there is no guarantee tha plane will be an active constraint in the vicinity of the optimum $\mathbf{w}^*$, nor must the ne of the reduced problem improve the master objective. In fact, it often occurs that $F($ As a result, a lot of the selected cutting planes do not contribute to the approximati objective around the optimum which, in turn, increases the number of iterations.

To speed up the convergence of CPA, we propose a new method which we ca *cutting plane algorithm* (OCA). Unlike standard CPA, OCA aims at simultaneously master and reduced problems' F and F_t objective functions, respectively. In additio select cutting planes that have a higher chance of actively contributing to the appro master objective function F around the optimum $\mathbf{w}^*$. In particular, we propose the changes to CPA.

and $\delta(\cdot,\cdot)$ is the zero-one loss function. This is a sensible choice of risk, as a point can only be classified if its contribution to the summand exceeds 1. As a result, this risk function provides an upper-bound on the average training error (or empirical risk) $R_{emp}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\delta(h_{\mathbf{w}}(\mathbf{x}), y_i)$.

Vectors $\mathbf{w}_t$ and $\mathbf{w}_t^b$ are computed as in the binary case. The latter computation again boils down to minimizing $f(\mu) = F(\mu\mathbf{w}_t + (1-\mu)\mathbf{w}_{t-1}^b)$. In this case, $\partial f(\mu)$ has discontinuities at $n(k-1)$values of μ (recall $k = |\mathcal{Y}|$). Finding each of these values can be done in $\mathcal{O}(k^2)$ time, and sorting them can be done in $\mathcal{O}(nk\log nk)$ time, so the overall time complexity of the line search is $\mathcal{O}(nk^2 + nk\log nk)$.

III. Improving Upon OCA

Though the math behind completing the line search for an optimal value of μ is elegant, the superlinear asymptotic runtime is an immediate warning flag, as Joachims' methods scale linearly with n. It seems that this portion of the OCA leaves something to be desired. When looking to improve the OCA algorithm, our primary focus was on developing faster techniques for choosing a good value of μ.

Using the terminology presented in [2], the CPA presented in [1] uses $\mu = 1$ for each iteration. While this hastens each iteration by avoiding a line search, the effect is a notable increase in the number of iterations required. The thought which inspired our research is that perhaps finding good approximations for μ could be done substantially faster than the complete line search (thus reducing time spent on each iteration), while keeping the number of iterations from increasing as dramatically as results from simply setting $\mu = 1$.

A. The Grid Search Cutting Plane Algorithm (GCA)

Our initial grid-search cutting-plane algorithm, GCA, rather than solving analytically for the maximum of $f(\mu)$, computes its value at several points and uses these to make an intelligent choice. The method by which this choice is made is inspired by the grid search for optimal parameter values presented in [13]. Given a parameter $\beta \in (0, 0.5]$, GCA computes $f(m\beta)$ for $m = 0, 1, \dots \lfloor\frac{2}{\beta}\rfloor$. Then, a more refined search is conducted in the neighborhood of the best choice of $m\beta$, incrementing by β^2 at each step. Finally, a third search occurs in the most promising region identified by the second pass, incrementing by β^3 at each step. The final output is the value of μ among those checked which minimized $f(\mu)$. This process is detailed in Algorithm 1, and makes use of a function **makePass** and a global variable $bestval$.

It should be noted that evaluating $f(\mu)$ requires $\mathcal{O}(nk)$ time, and that for any choice of β, the number of times that $f(\mu)$ is evaluated is bounded by a constant independent of n, k, C, or ϵ. As a result, GCA reduces the asymptotic runtime of the line search from $\mathcal{O}(nk^2 + nk\log nk)$ to $\mathcal{O}(nk)$.

One might be concerned that GCA will get stuck in a local minimum which is far from the true optimal value of μ. Fortunately, this cannot occur. The shape of the function $f(\mu)$ is described extensively in [2] and [3]. For the purposes of this paper, it is enough to note that ∂f, though not continuous, increases monotonically. In other words, the concavity of f never changes: it is always concave-up. As a result, f has no extraneous local minima for the grid search to find.

```
double makePass($init$, $max$, $inc$)
  $\mu' \leftarrow \frac{init+max}{2}$
  for ($\mu \leftarrow init; \mu < max; \mu \leftarrow \mu + inc$) do
    if ($f(\mu) < bestval$) then
      $bestval \leftarrow f(\mu)$
      $\mu' \leftarrow \mu$
    end if
  end for
  return $\mu'$
```

Algorithm 1 Grid Search Algorithm (GCA)

```
$bestval \leftarrow \infty$
$\mu_1 \leftarrow$makePass(0,2,$\beta$)
$\mu_2 \leftarrow$makePass($\mu_1 - \frac{\beta}{2}, \mu_1 + \frac{\beta}{2}, \beta^2$)
$\mu_3 \leftarrow$makePass($\mu_2 - \frac{\beta^2}{2}, \mu_2 + \frac{\beta^2}{2}, \beta^3$)
return $\mu_3$
```

The computationally expensive part of the grid search is repeatedly computing $f(\mu)$. If this is done too frequently, the GCA grid search will run more slowly than the OCA line search. This was the motivation behind the three-tiered approach described above. Conducting a single pass at high granularity would require computing $f(\mu)$ too many times. Conducting a single pass at low granularity would alleviate this problem, but at the expense of having a very rough estimate for the optimal value of μ. Instead, GCA essentially "zooms in" on the most promising sections by conducting high-granularity searches only in regions where $f(\mu)$ is relatively low.

Unsurprisingly, the value of the parameter β has a significant effect on the efficiency of the grid search. For small values of β, $f(\mu)$ is computed too frequently, leading to poor performance. For large values of β, $f(\mu)$ is evaluated fewer times, but the precision of the search is worse, so the number of iterations required to converge is greater. We experimentally determined that setting $\beta = 0.2$ consistently proved to be a good compromise between these influences. Unfortunately, even for large values of β (such as 0.5), a naïve grid search evaluates $f(\mu)$ too many times with each iteration, causing the grid search to be slower than the OCA line search. At a glance, this seems to spell doom for using a grid-search to determine μ.

B. An Improved Grid Search (GCA2)

Fortunately, there are ways to get around this problem while maintaining the core idea of using a grid-search. While GCA neatly encapsulates the central idea behind the grid-search, it requires several modifications in order to outperform OCA. Accordingly, we developed a revised grid-search algorithm, GCA2, which makes two crucial changes to GCA. These modifications are displayed in Algorithm 2 (which makes use of a modified routine **makePass2**), and described in more detail below.

The first of these optimizations takes advantage of the concavity of f by truncating the grid search if $f(m\beta) <$

```
double makepass2($init$, $inc$)
  $\mu \leftarrow init$
  while ($f(\mu + inc) < f(\mu)$) do
    $\mu \leftarrow \mu + inc$
  end while
  return $\mu$
```

Algorithm 2 Modified Grid Search Algorithm (GCA2)

```
  if ($uncertainty \geq t_2$) then
      $\mu_1 \leftarrow$ makepass2($\mu_{prev} - 1, \beta$)
  else $\mu_1 \leftarrow \mu_{prev}$
  end if
  if ($uncertainty \geq t_1$) then
      $\mu_2 \leftarrow$ makepass2($\mu_1 - \frac{\beta}{2}, \beta^2$)
  else $\mu_2 \leftarrow \mu_{prev}$
  end if
  $\mu_3 \leftarrow$ makepass2($\mu_2 - \frac{\beta^2}{2}, \beta^3$)
  if ($|\mu_3 - \mu_{prev}| \leq h(uncertainty)$) then
      decrease $uncertainty$
  else increase $uncertainty$
  end if
  $\mu_{prev} \leftarrow \mu_3$
  return $\mu_3$
```

$f((m+1)\beta)$, as this indicates that the search has passed the optimal value of μ. The technique ensures that time is not wasted calculating values of f at points which are increasingly far from optimal, and is also applied during the second and third passes of the algorithm. Another advantage to this is that there is no longer the arbitrary restriction $\mu < 2$.

The second change takes advantage of the observation that after the first few iterations, the optimal value of μ, as calculated by the OCA line search, was generally quite low (certainly below 0.1). In other words, as the algorithm progressed, $\mathbf{w}_t^b$ was modified only slightly with each iteration. As a result, a full grid search (even truncated as described above), wasted too much time checking large values of μ. To avoid this, we introduced a new variable, `uncertainty`, which essentially determines the width of the search window for each iteration.

Initially, `uncertainty` is set at 1, and all three passes of the line search are conducted. From then on, each iteration searches values of μ in an interval close the value of μ selected by the previous iteration. The width of this interval is determined by the value of `uncertainty`. There are two threshold values $t_1 < t_2$. If `uncertainty` $> t_2$, three passes are conducted. If $t_1 <$ `uncertainty` $< t_2$, the first (coarsest) search is bypassed, narrowing the search window. If `uncertainty` $< t_1$, only the final (most refined) search is conducted. After each iteration, the value of `uncertainty` is adjusted according to how much the optimal value of μ has changed from the previous iteration to the current one. If the difference between these is less than a threshold value h, `uncertainty` is decreased. Otherwise, it seems possible that a broader search was needed, so `uncertainty` is increased. It is best to make h depend on the width of the search region (which in turn is a function of `uncertainty`).

This algorithm takes advantage of the fact that most late iterations only modify $\mathbf{w}_t^b$ slightly by saving time when `uncertainty` is low, while still allowing the flexibility of a full grid search when necessary. A summary of the performance of GCA2 is presented in the following section, with more complete data provided in the Appendix.

C. The Three-point Cutting Plane Algorithm (TCA)

Inspired by previous work and hoping to take advantage of the shape of $f(\mu)$, we developed a second way to approximate optimal values of μ, which we refer to as the Three-point Cutting Plance Algorithm (TCA). This algorithm makes use of three possibilities for μ, designated `low`, `mid`, and `high`. These values define a search window which is initially centered at the value of μ selected by the previous iteration. The width of the initial window is determined by the variable `uncertainty`. If $f(\texttt{mid})$ is less than both $f(\texttt{low})$ and $f(\texttt{high})$, this indicates that the optimal value of μ lies within the current search region, so the search region is narrowed by shifting the values `low` and `high` towards `mid`. If $f(\texttt{low})$ is smaller than the other values, then our current search region is to the "right" of the optimal value of μ, so the entire window is shifted left. Similarly, if $f(\texttt{high})$ is the smallest value, the search region is shifted right. This continues until the width of the window, $\texttt{high}-\texttt{low}$, is less than a prescribed parameter γ, at which point we set $\mu = \texttt{mid}$.

When designing the algorithm, two aspects of it were tweaked to determine what provided the best performance. The first was, unsurprisingly, the parameter γ. The second was the amount by which `low` and `high` are shifted inwards when $f(\texttt{mid})$ is the lowest value. In general, the values were computed to be a weighted average of their previous values and `mid`. As is displayed in Algorithm 3, the relative weights of these points, α_l and α_h, are computed using a function g which takes as input $f(\texttt{mid})$ and $f(\texttt{low})$ or $f(\texttt{high})$, respectively. Intuitively, the closer $f(\texttt{low})$ is to $f(\texttt{mid})$, the less the point `low` should move, and so the larger the weight for α_l. After some experimentation, it was determined that $\gamma = 0.02$, $g(x,y) = (\frac{x}{y})^2$ provided good performance. Our implementation uses the equation $h(x) = x/2$ to determine whether to increase or decrease `uncertainty`.

IV. Experimental Results

Tests were primarily run on two data sets, both available from the LIBSVM data sets website (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/). The first, Covtype, provided cartographic data on sections of forest, and classified each location based on the dominant form of tree cover. The training set consists of 581012 examples, each with 54 features, divided into 7 classes. All features were linearly scaled to take values in the range [0,1]. The second data set, MNIST, presents a hand-written digit recognition problem. The training set consists of 60000 examples, each with 780 features, divided into 10 classes. An additional data set, RCV1, which uses data from the Reuters corpus and was also obtained from the LIBSVM website,

Algorithm 3 Three-point Cutting Plane Algorithm (TCA)

```
$low \leftarrow \mu_{prev} - uncertainty$
$mid \leftarrow \mu_{prev}$
$high \leftarrow \mu_{prev} + uncertainty$
while $(high - low > \gamma)$ do
    if $(f(low) < f(mid))$ then                /* shift search left */
        $high \leftarrow mid$
        $mid \leftarrow low$
        $low \leftarrow 2 \cdot mid - high$
    else if $(f(high) < f(mid))$ then /* shift search right */
        $low \leftarrow mid$
        $mid \leftarrow high$
        $high \leftarrow 2 \cdot low - mid$
    else                                  /* contract search window */
        $\alpha_l \leftarrow g(f(mid), f(low))$
        $\alpha_h \leftarrow g(f(mid), f(high))$
        $low \leftarrow \frac{mid+\alpha_l \cdot low}{1+\alpha_l}$
        $high \leftarrow \frac{mid+\alpha_h \cdot high}{1+\alpha_h}$
    end if
end while
if $(|mid - \mu_{prev}| \leq h(uncertainty))$ then
    decrease $uncertainty$
else increase $uncertainty$
end if
$\mu_{prev} \leftarrow mid$
return $mid$
```

was used for some experiments. Its test set (which we used for training) contains 518571 examples, each consisting of 47236 features. These examples are divided into 53 classes. Because the focus of this project is on data sets which require significant training time, these data sets were selected from the LIBSVM site primarily because they were among the largest. All tests were run on a Linux VirtualBox machine installed on a 2.8 GHz Intel Core i7 processor with 8 GB main memory.

A. Results Using Default Parameter Values

The graphs in Figures 2 and 3 show results when using the default parameter values of $C = 1, \epsilon = 0.01$. Search Time represents the total time spent determining which value of μ to use for each iteration. All tests reached the termination criteria $1 - \frac{F_t(\mathbf{w}_t^b)}{F(\mathbf{w}_t)} < \epsilon$, and classification performance for the three models was equivalent (OCA, GCA2 and TCA error rates were 35.92%, 35.90%, and 35.84%, respectively, on Covtype, and 5.57%, 5.56%, and 5.56% on MNIST). Data from more experiments, along with discussion of the effect of changing these parameters, follows in sections IV-B, IV-C, and the Appendix.

The algorithms GCA2 and TCA notably outperform OCA on both of the primary data sets. On MNIST, the total time spent finding a good value for μ decreased by over 70% for both algorithms, causing the time spent per iteration to decrease by just over 16%. As the number of iterations increased by only 2-3%, the net effect was approximately a 14% improvement in training time. On Covtype, results were even better: both algorithms take over 38% less time to train than OCA. This is achieved by reducing the time spent selecting μ by over 65-70% while increasing the number of iterations by less than 10%.

It is not surprising that GCA2 and TCA perform relatively better on Covtype than on MNIST, as the OCA line search consumes over 56% of the total time on the former, and only 21% of the total time on the latter. When tested on RCV1 the three algorithms performed equivalently (slight fluctuations from one trial to the next caused variation in which algorithm terminated first). It is unsurprising that GCA2 and TCA do not improve upon OCA in this case, as under 8% of OCA training time was spent on the line search. It is natural to ask what causes this discrepancy, since understanding it would make it possible to determine (without explicitly running tests) whether or not the algorithms presented in this paper were likely to provide a significant reduction in training time on a particular data set.

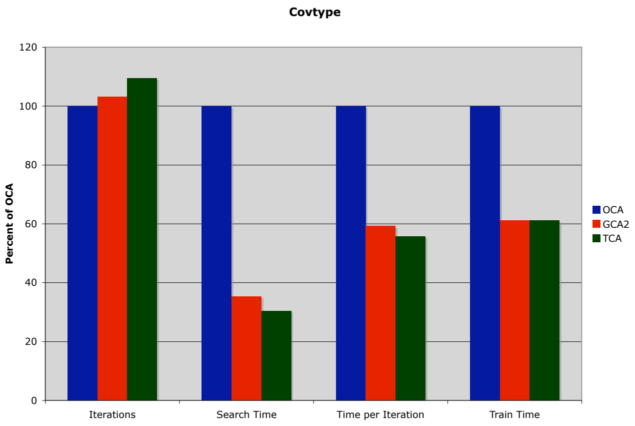


Fig. 2. Performance on Covtype, normalized to OCA performance. Note that search time, the focus of GCA2 and TCA, is reduced by 60-70%, causing an improvement in time of about 40%.

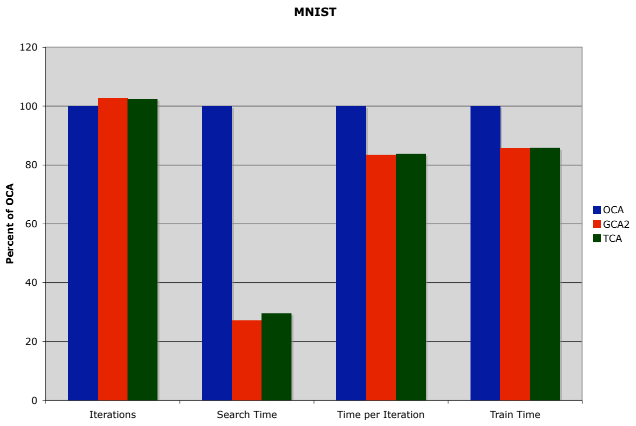


Fig. 3. Performance on MNIST, normalized to OCA performance. Note that despite improving search time by over 70%, GCA2 and TCA training times do not decrease as substantially as for Covtype.

B. Examining the Number of Features

It should be noted that RCV1, for which the line search consumed only 7.8% of the total time, has over 47000 features. On MNIST, which makes use of 780 features, 21% of the total time was devoted to the line search. Covtype, meanwhile, uses only 54 features, and the line search consumes 56% of the total time. This pattern seemed worth examining more closely.

From a theoretical perspective, the OCA line search consists of computing $\mathcal{O}(nk)$ values of μ where the gradient of $f(\mu)$ might have a discontinuity, and then sorting these values. While time spent on the initial computation depends on the number of features used, the line search time is dominated by the sorting of the values [2], which takes time dependent only on n. Accordingly, while time spent on the line search is not technically independent of the number of features s, for data sets with a large number of training examples (which are the only ones of relevance for this paper), the time spent on the line search should not be greatly affected by the number of features. Other portions of the algorithm, however, take time scaling linearly with s. As a result, the fraction of training time spent on the line search should increase as the number of features decreases.

In order to test this hypothesis, we created new training sets from the RCV1 test set and MNIST training set. These training sets contained all of the original data points, but left out many of the original features. We then ran the OCAS solver on each of these new data sets. The results, using $C = 10, \epsilon = 0.01$ for MNIST and $C = 1, \epsilon = 0.01$ for RCV1, are shown in Figure 4.

As the number of features decreases, the total time spent per iteration decreases, while the time spent during each iteration on the line search remains fairly constant. As a result, the percentage of the total time spent on the line search increases notably. Because GCA2 and TCA focus on improving the OCA line search, this indicates that these methods will outperform OCA by the most on data sets with relatively few features.

Of course, one cannot arbitrarily reduce the dimensionality of a data set without potentially reducing classification performance. Because decreasing numbers of features are accompanied by decreasing training time, there is already a notable incentive to conduct research into how best to identify and remove less informative features early in the classification process. This paper does not seek to develop techniques to accomplish this. Instead, we stop at noting which data sets are most impacted by GCA2 and TCA modifications, and observing that if techniques for the elimination of uninformative features improve, the algorithms presented here will become relatively more advantageous.

Features	% Search	Features	% Search
780	18.8	4373	30.0
473	25.1	2368	31.9
337	36.7	676	42.5
139	57.2	186	56.4

Fig. 4. For both MNIST (left) and RCV1 (right), as the number of features increases, the percentage of training time spent on the OCA line search increases.

It seems likely that a more thorough examination of this phenomena, rather than looking at the raw number of features s for a data set, should consider how s compares with the number of training examples n, as this value crucially determines run-times for various parts of the algorithm. We leave this for future research.

C. Effects of Parameters C and ϵ

It should be noted that the percentage of the training time spent on the line search depends not only on the data, but also on the parameters used when solving.

If the tolerance ϵ is very small, then the program will have to go through more iterations to obtain the desired accuracy. Because the set of constraints grows with the number of iterations, the amount of time required to solve these constraints (referred to as Quadratic Programming, or QP, Time) grows super-linearly with respect to the number of iterations. The other steps of the algorithm, including the line search, require a constant amount of time per iteration. This is shown in Figure 5. As a result, as ϵ decreases, QP time comes to dominate the training time, and so the percentage of the process spent on the line search decreases. Accordingly, methods which aim to decrease training time by focusing on the line search, such as GCA2 and TCA, are less likely to demonstrate a notable reduction in training time.

At some point, any additional iterations required as a result of sub-optimal determination of μ should outweigh the reduced time spent on the line search, causing OCA to outperform GCA2 and TCA. It should be noted, however, that classification accuracy universally converges to its final value for values of ϵ much greater than this threshold value. Correspondingly, there is no need for the user of the program to set ϵ to values low enough to cause this behavior.

Unsurprisingly, given the bound of $\mathcal{O}(C/\epsilon)$ on the number of iterations, increasing C has a similar effect to decreasing ϵ. Thus, for sufficiently large values of C, time spent solving the

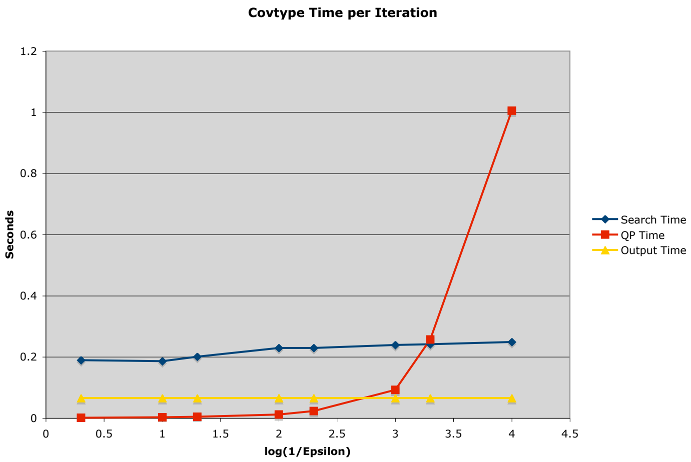


Fig. 5. Covtype Time Breakdown. As ϵ decreases, the number of iterations required to converge increases. While most steps in the algorithm take a constant amount of time per iteration, this causes time spent solving the reduced optimization problem, QP Time, to increase dramatically.

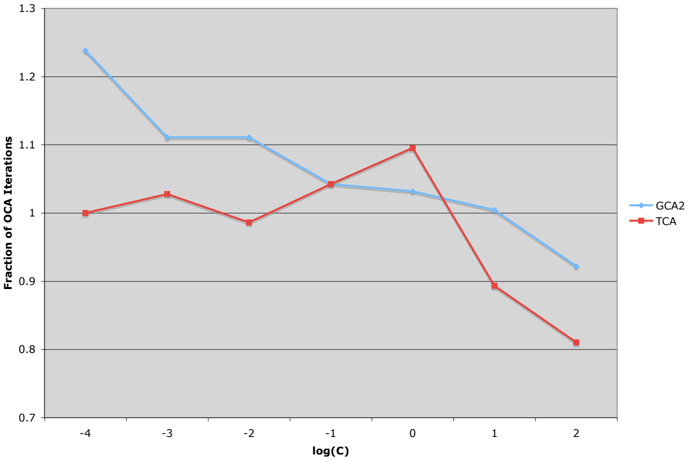


Fig. 6. Effect of C on Iterations Ratio. As the regularization constant C increased, the ratio of iterations required by GCA2 and TCA to the number required by OCA decreased. Thus, even as search time becomes a negligible fraction of training time, these new algorithms continue to outperform OCA.

quadratic programming problem dwarfs all other components. Interestingly, however, increasing C sometimes has another effect on the training times for the three algorithms being discussed. On the Covtype data set, as C grows very large, OCA becomes increasingly likely to use $\mu = 0$ for many consecutive iterations. For reasons described in the following section, this frequently results in GCA2 and TCA terminating in fewer iterations than OCA. Thus, even for values of C where time spent on the line search comprises a negligible fraction of the total time, GCA2 and TCA may improve upon the the performance of OCA. Figure 6 displays the relative number of iterations for GCA2 and TCA (when compared to OCA) for different values of C (logarithm for x-axis is base 10). Tables showing the full results of tests run with different values of C and ϵ are provided in the Appendix.

V. Another Benefit to GCA2 and TCA

When seeking an approximate solution to the primal optimization problem (1), both CPA and OCA use a greedy approach: each iteration attempts to reduce the objective value by as much as possible. Since the ultimate goal is to find an objective value within a tolerance ϵ of the optimal value, this is a reasonable approach. It should be noted, however, that choices for $\mathbf{w}_t^b$ have a notable impact on values of $\mathbf{w}_i^b$ for $i > t$. This means that a slightly "worse" choice of $\mathbf{w}_t^b$ during the current iteration may result in better options in the future, and lead to faster convergence.

Recall that in OCA, $\mathbf{w}_t^b$ is found by searching along a line between $\mathbf{w}_{t-1}^b$ and $\mathbf{w}_t$. Thus, the following section will discuss finding optimal values of μ (rather than $\mathbf{w}_t^b$), as the two are equivalent.

The inspiration for GCA2 and TCA was that by approximating the optimal value of μ, rather than finding it exactly, the time spent on each iteration of the algorithm would decrease. It was assumed that there was a tradeoff: in return for reducing the time per iteration, the values of μ that were found would not be as good as those found with OCA, and as a result the above two algorithms would require more iterations to converge. While this is often the case, for a notable number of experiments, GCA2 and TCA terminated after *fewer* iterations than OCA (in addition to requiring less time per iteration). This is only possible because of the observation that all of the algorithms presented in this paper use a greedy solution to a problem which is not inherently greedy.

The above observation is only interesting if it is possible to identify values of μ which will be particularly good or bad for future convergence. In general, this is difficult. Nevertheless, there is one value of μ which has a clear drawback. Note that if $\mu = 0$, $\mathbf{w}_t^b = \mathbf{w}_{t-1}^b$. While it is true that $\mathbf{w}_t$ and $\mathbf{w}_{t+1}$ will be different (so the algorithm will never truly get "stuck"), if $\mu = 0$ for many iterations in a row, the algorithm has spent all of those iterations without improving its best so-far solution.

Perhaps the data set which best illustrates this danger is "Poker," which can be obtained from the UCI machine learning repository of data bases (http://archive.ics.uci.edu/ml/datasets.html). It contains eight million example 5-card poker hands, with ten features used to describe each (the suit and rank of each card). There are 10 class labels, which describe the quality of the hand (high card, pair, two pairs, etc). When run on this data set with the default values of $C = 1, \epsilon = 0.01$ TCA and GCA2 converged in 234 and 235 iterations, respectively, while OCA required 1454 iterations and took over twelve times as long to find an equivalent solution. Upon further investigation, it became apparent that the concern noted above (namely, repeatedly determining $\mu = 0$ to be optimal) was precisely what caused OCA to perform so poorly. For 1451 consecutive iterations of OCA, $\mu = 0$ was computed to be optimal. Thus, $\mathbf{w}_t^b$ remained at its default value, $\mathbf{0}$, and classification error was 100%. Because of the way that GCA2 and TCA are implemented, these algorithms tended to find small (but nonzero) values for μ, and thus avoided the trap into which OCA fell.

Interestingly, this problem can be avoided by a very simple fix. After modifying OCA to arbitrarily set $\mu = 0.02$ for the first five iterations (after which it returned to its usual line search), the algorithm terminated in 231 iterations, with comparable objective value and classification performance to GCA2 and TCA.

The fact that for $\epsilon > .011$, OCA terminated without ever modifying $\mathbf{w}$ serves as a potent reminder that while solving the primal problem and achieving accurate classification score are often linked (and most SVMs are compared on the basis of their respective objective values), it is possible to succeed at the first without making notable progress on the second.

It should be noted that for many reasons, the Poker data set is not well-suited to multi-class SVM classification. With so few features relative to the number of examples, the SVM is unable to learn a good model - the best it can do is to get about 50% correct. Even this is unimpressive, as over 92% of the examples belong to two of the classes, and so the final predictions are no better than simply assigning the most common label to every hand.

We argue that the limitations of the Poker data set should not undermine the validity of the observations above. The first reason for this is that even if it is not possible to achieve high classification accuracy, one would hope to achieve the best result possible in relatively few iterations. The second is that while Poker was selected because it provides a particularly glaring example of this concern, the same issue arises when working with other data sets.

To prove this point, we examined the trials run on the Covtype data set, with $C = 100, \epsilon = 0.01$. In this case, OCA requires 744 iterations to converge, while GCA2 and TCA require 686 and 603, respectively. Upon investigation, it was determined that on 490 of the 744 OCA iterations (or 66%), the optimal value of μ was determined to be 0. After modifying the code so that any time $\mu = 0$ was determined to be optimal, μ was instead set to 0.02, OCA terminated in 569 iterations.

These results indicate that the shortcomings of the greedy approaches used by all cutting plane algorithms described in this paper may, in some cases, be quite significant. The quick fixes mentioned here may not be the best way to avoid these concerns, and an investigation into better ways to approach this problem would be fascinating and potentially very useful. Such an exploration, however, is beyond the scope of this paper. For our purposes, it suffices to note that while setting $\mu = 0$ may not always be problematic, repeatedly choosing doing so can result in poor performance.

VI. Conclusion and Future Work

In this paper, we have observed that the line search used to compute the best-so-far solution $\mathbf{w}_t^b$ at each iteration of the OCA algorithm is fairly computationally costly. We present two new algorithms, GCA2 and TCA, designed to approximate this solution in a substantially shorter amount of time. These solutions reduce the asymptotic run-time for the search from $\mathcal{O}(nk^2 + nk \log nk)$ to $\mathcal{O}(nk)$, and demonstrate a notable decrease in training time when tested empirically. Both algorithms reduce the search time by 65-75% while seeing an increase in the number of iterations of approximately 3%. On the two primary data sets used for our tests, Covtype and MNIST, the net effect was a decrease training time by roughly 39% and 14%, respectively.

Additionally, we have examined and analyzed the ways in which the regularization constant C, the tolerance ϵ, and the number of features in the training set impact the performance of GCA2 and TCA relative to that of OCA. For reasonable choices of C and ϵ, the number of features present in a data set was shown to significantly impact the percentage of OCA training time spent on the line search: search time was relatively independent of the number of features, while other portions of the algorithm became significantly slower on data sets with large numbers of features. As a consequence of this fact, GCA2 and TCA are most beneficial on large data sets with relatively few features.

Perhaps one of the most interesting observations was that while GCA2 and TCA do not find the optimal choice of $\mathbf{w}_t^b$ at each iteration, for some data sets they converge in *fewer* iterations than OCA, occasionally by quite a significant margin. This suggests that the approach of reducing the objective function by as much as possible at each iteration can yield far-from-optimal rates of convergence. An open question for future investigation is *which* greedy choices are problematic and whether there are better methods for choosing $\mathbf{w}_t^b$ than the approaches that have so far been employed.

Finally, it is worth noting that while the experiments used for this paper were restricted to multi-class classification tasks, the conceptual ideas presented in this paper apply equally well to binary and multi-class problems. It would be informative to conduct tests on binary data sets, to see whether empirical results are are similar in the binary and multi-class cases. In addition to observing whether GCA2 and TCA cause training times to decrease by comparable ratios to those reported in this paper, we would like to investigate whether the cases where GCA2 and TCA terminate in fewer iterations than OCA are more or less prevalent when solving binary classification problems.

Acknowledgement

This research was conducted with support from NSF grant ARRA 0851783. The author would like to thank Dr. Jugal Kalita for pointing the way to relevant recent publications and providing valuable feedback during the editing process.

References

[1] T. Joachims, "Training Linear SVMs in Linear Time," in *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM New York, NY, USA, 2006, pp. 217-226.

[2] V. Franc and S. Sonnenburg, "OCAS Optimized Cutting Plane Algorithm for Support Vector Machines", in *Proceedings of International Machine Learning Conference*, pages 320327. ACM Press, 2008a.

[3] V. Franc and S. Sonnenburg, "Optimized Cutting Plane Algorithm for Large-Scale Risk Minimization", in *Jounal of Machine Learning Research*, October 2009.

[4] E. Mayoraz and E. Alpaydin, "Support Vector Machines for Multi-Class Classification", in *IWANN, vol. 2*, 1999, pp. 833-842.

[5] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," in *Data Mining and Knowledge Discovery*. Kluwer Academic Publishers, Bowston, MA, USA, 1998, pp. 121-167.

[6] C. Hsu and C. Lin, "Comparison of Methods for Multiclass Support Vector Machines" in *IEEE Transactions on Neural Networks*, vol. 13, no. 2, March 2002.

[7] H. Lei and V. Govindaraju, "Half-Against-Half Multi-class Support Vector Machines," in *Proc. of the 6th International Workshop on Multiple Classifier Systems*, Seaside, CA, USA, 2005.

[8] T. Dietterich and G. Bakiri, "Solving Multiclass Learning Problems via Error-Correcting Output Codes," in *Journal of Artificial Intelligence Research 2*, 1995.

[9] K. Crammer and Y. Singer, "On the Algorithmic Implementation of Multiclass Kernel-Based Vector Machines" in *The Journal of Machine Learning Research*, vol. 2, 2002, pp. 265-292.

[10] J. Weston and C. Watkins, "Support Vector Machines for Multi-Class Pattern Recognition", in *European Symposium on Artificial Neural Networks*. D-Facto public, Belgium, 1999, pp. 219-224.

[11] S. Keerthi, S. Sundararajan, K Change, C. Hsieh, and C. Lin, "A Sequential Dual Method for Large Scale Multi-Class Linear SVMs" in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, August 24-27, 2008, Las Vegas, Nevada, USA.

[12] T. Joachims, T. Finley, and C.N. Yu, "Cutting-plane training of structural SVMs" in *Machine Learning*, 76(1), May 2009.

[13] C. Hsu, C. Chang, and C. Lin, "A Practical Guide to Support Vector Classification." Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 2003. http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

Streaming Trend Detection in Twitter

James Benhardus

***Abstract*—Twitter is a popular microblogging and social networking service with over 100 million users. Users create short messages pertaining to a wide variety of topics. Certain topics are highlighted by Twitter as the most popular and are known as "trending topics." In this paper, we will outline methodologies of detecting and identifying trending topics from streaming data. Data from Twitter's streaming API will be collected and put into documents of equal duration. Data collection procedures will allow for analysis over multiple timespans, including those not currently associated with Twitter-identified trending topics. Term frequency-inverse document frequency analysis and relative normalized term frequency analysis are performed on the documents to identify the trending topics. Relative normalized term frequency analysis identifies unigrams, bigrams, and trigrams as trending topics, while term frequcny-inverse document frequency analysis identifies unigrams as trending topics.**

***Index Terms*—microblogs, trend detection, natural language processing**

I. Introduction

TWITTER is a popular microblogging and social networking service that presents many opportunities for research in natural language processing (NLP) and machine learning. Since its inception in 2006, Twitter has grown to the point where http://twitter.com is the 11th most visited website in the world, and the 8th most visited site in the United States[1], and over 100 million Twitter accounts have been created [2]. Users of Twitter post short (less than than or equal to 140 character) messages, called "tweets," on a variety of topics, ranging from news events and pop culture, to mundane daily events and spam postings. As of February 2010, users of Twitter were producing 50 million tweets per day, an average of 600 tweets per second[3].

Twitter presents some intriguing opportunites for applications of NLP and machine learning. One such aspect of Twitter that provides opportunities is trending topics - words and phrases, highlighted on the main page of Twitter, that are currently popular in users' tweets. Trending topics are identified for the past hour, day and week. Examples of trending topics can be seen in Fig. 1 and Fig. 2. Trending topics are supposed to represent the popular "topics of conversation," so to speak, among the users of Twitter. Determining trending topics can be considered a type of First Story Detection (FSD), a subset of the larger problem known as Topic Detection and Tracking (TDT) [1]. The popularity and growth of Twitter presents some challenges for applications of NLP and machine learning, however. The length restrictions of the messages create syntactical and structural conventions that are not seen in more traditional corpora, and the size of the Twitter network produces a continuously changing, dynamic corpus. In addition, there is quite a lot of content on Twitter that would be classified as unimportant to an outside observer, consisting of personal information or spam, which must be filtered out in order to accurately identify the elements of the corpus that are relevant to the Twitter community as a whole, and could thus be considered to be potential trending topics. The challenge of Twitter's popularity is that in order to detect and identify trending topics, one must sample and analyze a large volume of streaming data. This paper will propose methods of using natural language processing techniques on streaming data from Twitter to identify trending topics.

II. Related Work

While there is a large body of work pertaining to natural language processing, applying NLP techniques to Twitter is a fairly recent development, due in part to the fact that Twitter has only been in existence since 2006[4]. In this relatively short span of time, however, there have been many insightful analyses of Twitter. In particular, there are several recent applications natural language processing techniques to Twitter:

- Twitter has been used to study the dynamics of social networks, particularly the temporal behavior of social networks [11], or the behavior of social neworks during disasters, such as earthquakes [8], [14].
- First story detection has been applied to Twitter to identify the first tweets to mention a particular event [12].
- Data mining from trending topics have also been applied to Twitter to summarize trending topics [17] and to analyze how trending topics change over time [2].
- In addition to applications of NLP techniques to Twitter, trend and event detection techniques have also been applied to other online entities such as weblogs [3], [4], news stories [1], [10], [20], or scientific journal collections [16], [19].

III. Problem Definition

The main goal of this project is to detect and identify trending topics from streaming Twitter data. To accurately define the problem, the first step must be to define explicitly what constitutes a trending topic. In [4], topics are defined as consisting of a combination of *chatter*, which is characterized by persistent discussion at a constant level and is largely

J. Benhardus (Physics Department - Bethel University, St. Paul, MN 55112) is a rising senior participating in a Summer 2010 National Science Foundation REU for Artificial Intelligence, Natural Language Processing and Information Retrieval at the University of Colorado at Colorado Springs, Colorado Springs, CO, 80918.

email: benjam@bethel.edu

[1] http://www.alexa.com/siteinfo/twitter.com

[2] http://economictimes.indiatimes.com/infotech/internet/Twitter-snags-over-100-million-users-eyes-money-making/articleshow/5808927.cms

[3] http://blog.twitter.com/2010/02/measuring-tweets.html

[4] http://en.wikipedia.org/wiki/Twitter

user-initiated, and *spikes*, which are characterized by short-term, high intensity discussion that is often in response to a recent event. In general, trending topics consist mainly of spikes. However, trending topics can also consist of a fairly even combination of spikes and chatter, or of mainly chatter. Examples from Fig. 1 and Fig. 2 of trending topics that could be considered to consist mainly of spikes are:

- CALA BOCA GALVAO
- Gonzalo Higuain
- Sani Kaita
- FIFA World Cup
- #worldcup
- Grecia
- Maradona
- Vuvuzela
- Oil Spill
- Tony Hayward
- Oswalt
- Shirley Sherrod
- Breitbart

Examples from Fig. 1 and Fig. 2 of trending topics that could be considered to be a fairly even combination of spikes and chatter are:

- #theview
- Jersey Shore tonight
- Thor

Examples from Fig. 1 and Fig. 2 of trending topics that could be considered too consist mainly of chatter are:

- Inception
- #iconfess
- #dontcountonit
- BUSTIN DREW JIEBER

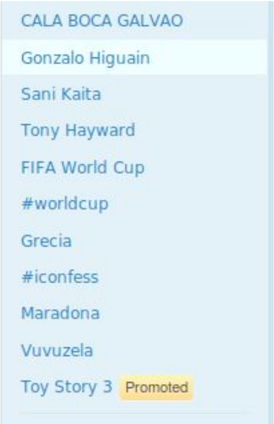
The “#” symbol at the beginning of “#worldcup”, “#iconfess”, “#theview”, and “#dontcountonit” is called a “hashtag,” and is used by Twitter users to classify their tweets as pertaining to a particular topic. In addition to spikes and chatter, a trending topic can also be the result of advertisement, as is the case for the final trending topic in Fig. 1, “Toy Story 3.” In this third possibility, the trending topic is associated with a “Promoted” tweet - a hybrid tweet-advertisement which is displayed at the top of search results on relevant topics[5].

While the classification of a trending topic as consisting of spikes or chatter is helpful for the understanding of the nature of trending topics, it is not directly useful in the identification or classification of terms as trending topics. Our working definition of a trending topic shall be a word or phrase that is experiencing an increase in usage, both in relation to its long-term usage and in relation to the usage of other words. More techincal definitions of trending topics shall be used in the actual experiments, and shall be described in the “Methodologies” section.

In addition to defining what constitutes a trending topic, we must also define what constitutes success for a particular methodology. As the goal of the project is to develop a method of identifying trending topics that is independent of the method used by Twitter, simple agreement with the Twitter-identified trending topics is both unambitious and potentially unrealistic without replicating Twitter’s methodology, which happens to be proprietary. As such, we shall define a successful method as a method that produces relevant topics at a rate of at least 75% of the rate returned by Twitter’s method, with an agreement of at least 50% with the terms produced by Twitter’s method. The details of computing relevance and agreement shall be discussed in the “Evaluation Measures” section.

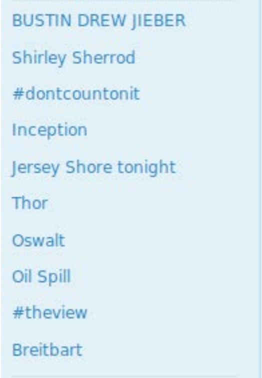
IV. Methodologies

Multiple methodologies were implemented, making use of one or more selection criteria. Each selection criterion will be discussed in its own subsection. All methods implemented made use both of the Twitter Streaming API[6] and the Edinburgh Twitter corpus [13], a collection of approximately 97 million tweets collected between November 2009 and February 2010. The Edinburgh Twitter corpus was used to provide baseline measurement against the data from the Twitter Streaming API. For each source, tweets were temporally



CALA BOCA GALVAO
Gonzalo Higuain
Sani Kaita
Tony Hayward
FIFA World Cup
#worldcup
Grecia
#iconfess
Maradona
Vuvuzela
Toy Story 3 Promoted

Fig. 1: A list of trending topics as identified by Twitter from 17 June 2010.
(Source: http://twitter.com)



BUSTIN DREW JIEBER
Shirley Sherrod
#dontcountonit
Inception
Jersey Shore tonight
Thor
Oswalt
Oil Spill
#theview
Breitbart

Fig. 2: A list of trending topics as identified by Twitter from 29 July 2010.
(Source: http://twitter.com)

[5] http://blog.twitter.com/2010/04/hello-world.html

[6] http://stream.twitter.com/1/statuses/sample.json (see documentation at http://apiwiki.twitter.com/Streaming-API-Documentation)

grouped into "bag of words" style collections, or "documents." These documents were be normalized by duration, meaning that each document corresponds to the tweets posted in a certain constant length of time. The Edinburgh Twitter corpus was divided into 1212 sections, each consisting of one hour's worth of tweets. The tweets from the Twitter Streaming API were grouped into sections corresponding to either ten minutes, fifteen minutes, or one hour's worth of data collection.

A. Frequency

The first criterion used was simply the raw frequency of each term. This criterion was used mainly as a threshold criterion, to be used with one or more of the other criteria. Using raw frequency by itself has major drawbacks, as the most frequent terms in the stream are the terms that carry the least information, such as "the", "and", or "rt" (an abbreviation for "retweet," a term used when one Tiwtter user reposts another Twitter user's tweet). The majority of the most common words can be classified as stop words, and filtered out of the stream. Generation of a stop word list shall be discussed in the "Experiments" section.

B. TF-IDF

The second criterion implemented involved analyzing each document using an application of tf-idf weighting. Tf-idf weighting is a information retrevial technique that weights a document's relevance to a query based on a composite of the query's *term frequency* and *inverse document frequency* [15]. Term frequency can be defined as either

$$tf_{i,j} = n_{i,j}$$

or

$$tf_{i,j} = \frac{n_{i,j}}{N}$$

where $n_{i,j}$ is the number of times word i occurs in document j and

$$N = \sum_k n_{k,j}$$

is the total number of words in document j. The second definition of $tf_{i,j}$ is often referred to as the *normalized term frequency*. Inverse document frequency is defined as

$$idf_i = log(\frac{D}{d_i})$$

where d_i is the number of documents that contain word i and D is the total number of documents. Put simply, the weight of a document will be higher if the number of times a word occurs in a document is higher, or if the number of documents containing that word is lower; similarly, the weight of a document will be lower if the number of times a word occurs in a document is lower, or if the number of documents containing that word is higher [5].

C. Normalized Term Frequency

The third criterion implemented involved utilizing only the term frequency of each element, rather than both the term frequency and the inverse document frequency. For this method, a *normalized term frequency* was used, defined as

$$tf_{norm_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}} * 10^6$$

where n_i is the number of times word i occurs in document j and $\sum_k n_{k,j}$ is the total number of words in document j. Due to the large number of words found in the documents, a scaling factor of 10^6 was used, meaning $tf_{norm_{i,j}}$ can be thought in terms of frequency per million words. Each word in the test document was given a *trending score*, defined as

$$ts_{i,j} = \frac{tf_{norm_{i,j}}}{atf_{norm_{i,S}}}$$

in which

$$atf_{norm_{i,S}} = \sum_{S=\{s_1,\ldots,s_p\}} \frac{tf_{norm_{i,s_k}}}{p}$$

where S is the set of p baseline documents to which the test document was compared.

D. Entropy

The fourth criterion implemented was entropy. To calculate the entropy of a term, all of the tweets containing that term are collected. As it is used in this project, the entropy of a term i is defined as

$$H_i = -\sum_j \frac{n_{j,i}}{N} log(\frac{n_{j,i}}{N})$$

where $n_{j,i}$ is the number of times word j occurs in the collection of tweets containing term i and

$$N = \sum_j n_{j,i}$$

is the total number of words in the collection of tweets containing term i. Entropy proved to be a helpful parameter to use in filtering out terms that could be classified as spam.

V. Experiments

Two experiments were run, implementing slightly different methodologies, but following the same general format. Unless stated otherwise, the process described was used for both experiments.

A. Data Collection

Data was collected using the Twitter streaming API, with the gardenhose tweet stream providing the input data and the trends/location stream providing the list of terms identified by Twitter as trending topics. The gardenhose streaming API is a limited stream that returns approximately 15% of all Twitter activity[7]. The trends/location stream provides a list of trending topics that is specific to a particular location. The United States

[7] http://dev.twitter.com/pages/streaming_api_concepts

was used as the location for evaluation, as both experimental methods worked almost entirely with English tweets, and most of the trending topics from the United States were in English, leading to a more accurate basis for comparison than trending topics from the entire world. The streaming data was collected automatically using the cURL data transfer tool within a shell script. The streaming data was grouped into documents of equal duration. The first experiment used documents consisting of tweets collected over either ten minutes or one hour of streaming. The second experiment used documents consisting of tweets collected over fifteen minutes of streaming.

B. Preprocessing

The data was collected from the Twitter streaming API in JSON format and a parser to extract the tweets from the other information returned. Next the tweets were preprocessed to remove URL's, unicode characters, usernames, and punctuation. A dictionary containing approximately 180,000 common English words and acronyms was used to filter out tweets that did not contain at least 60% English words. Tweets were classified as spam and discarded if one word accounted for over half of the words in the tweet. After preprocessing, tweets were stored in two ways - in a collection in which each valid tweet was left intact, and in a "bag of words" style dictionary consisting of a unigram and the frequency of the unigram in the document.

C. Baseline Data

Baseline data was computed from the Edinburgh Twitter Corpus, a collection of over 97 million tweets collected over three months in late 2009 and early 2010. The corpus was divided into 1212 sections corresponding to one hour's worth of tweets, consisting of two bag-of-words dictionarys for each section - one containing unigrams and one containing bigrams. For the first experiment, the resulting documents were used independently of one another. For the second experiment, the documents were compiled into a comprehensive dictionary of 805,025 words with term frequency, document frequency, and tf-idf weights computed for each word. For the first experiment, a specified number of baseline documents was used to compute average normalized term frequency. For the second experiment, the dictionary was used to provide document frequencies for terms and for the generation of a stop word list.

D. Stop Words

For each experiment, a list of stop words was used as an additional filter after preprocessing. A stop word is defined as a word that contains no meaning or relevance in and of itself, or a word that adds to the relevance of a result to a query no more often than would a random word [18].

For the first experiment, stop word were identified using a "lossy counting" algorithm [9]. The lossy counting algorithm identified the most frequent words in each of the 1212 baseline documents. All words that appeared as the most frequent in at least 75% of the baseline documents were classified as stop words. If a word in the test data was identified as a stop word, it was immediately removed from consideration as a potential trending topic.

For the second experiment, a word was considered to be a stop word if it matched one or more of the following criteria:

- If the word appeared in over 600 of the 1212 documents
- If the word had a total frequency of at least 3000 throughout all 1212 documents
- If the word was classified grammatically as a preposition or a conjunction
- If the word was a derivative of a word that occurred in 1200 or more documents (i.e. "can" occurs in all 1212 documents, so "can't," "could," "couldn't," and "could've" are also classified as stop words)

As with the first experiment, if a word in the test data was identified as a stop word, it was immediately removed from consideration as a potential trending topic.

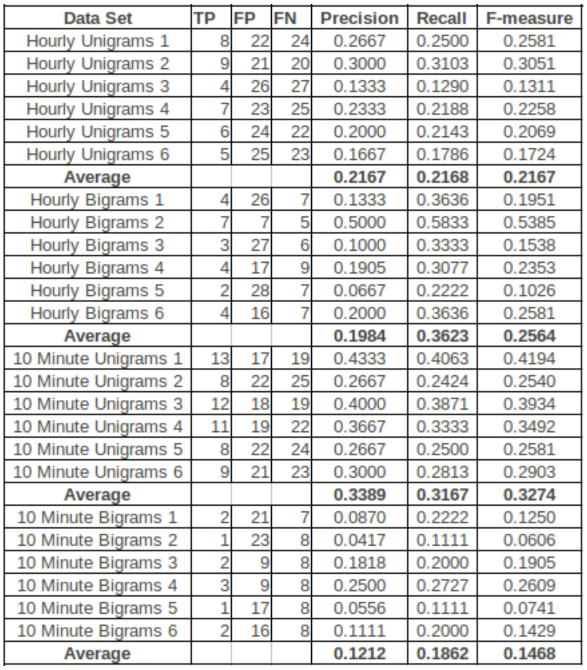
E. Selection Criteria

For the first experiment, a combination of raw frequency and relative normalized term frequency was used. The raw frequency was used as a threshold, eliminating all terms that did not occur an average of at least one time for every minute of data collection. Normalized term frequency and average normalized term frequency was calculated for each remaining term, and the terms with the highest trending scores were identified as trending topics. Analysis was performed for both unigrams and bigrams. Entropy was also calculated for both unigrams and bigrams, but was not used as a selection criterion for this experiment.

The second experiment utilized a combination of raw frequency, tf-idf weighting, and entropy to identify trending topics. Once again, the raw frequency was used as a threshold, eliminating all terms that did not occur an average of at least one time for every minute of data collection. Term frequency-inverse document frequency weights were calculated for the remaining terms. Of the remaining terms, those with a tf-idf weight below a threshold value (set at five greater than the length of data collection in minutes) were removed from consideration. Finally, terms with an entropy of less than 3.0 were removed, and the remaining terms were identified as trending topics.

VI. Evaluation Measures

The first experiment was evaluated using precision, recall, and F-measure scores in comparison to the trending topics identified by Twitter. All three measures require calculating the number of true positives - that is, the items that were identified as trending topics both by the experimental method and Twitter's method. In addition, determining precision requires calculating the number of false positives - the items identified as trending topics by the experimental method that were not identified as trending topics by Twitter, and determining recall requires calculating the number of false negatives - the items identified as trending topics by Twitter that were not identified



Data Set	TP	FP	FN	Precision	Recall	F-measure
Hourly Unigrams 1	8	22	24	0.2667	0.2500	0.2581
Hourly Unigrams 2	9	21	20	0.3000	0.3103	0.3051
Hourly Unigrams 3	4	26	27	0.1333	0.1290	0.1311
Hourly Unigrams 4	7	23	25	0.2333	0.2188	0.2258
Hourly Unigrams 5	6	24	22	0.2000	0.2143	0.2069
Hourly Unigrams 6	5	25	23	0.1667	0.1786	0.1724
Average				**0.2167**	**0.2168**	**0.2167**
Hourly Bigrams 1	4	26	7	0.1333	0.3636	0.1951
Hourly Bigrams 2	7	7	5	0.5000	0.5833	0.5385
Hourly Bigrams 3	3	27	6	0.1000	0.3333	0.1538
Hourly Bigrams 4	4	17	9	0.1905	0.3077	0.2353
Hourly Bigrams 5	2	28	7	0.0667	0.2222	0.1026
Hourly Bigrams 6	4	16	7	0.2000	0.3636	0.2581
Average				**0.1984**	**0.3623**	**0.2564**
10 Minute Unigrams 1	13	17	19	0.4333	0.4063	0.4194
10 Minute Unigrams 2	8	22	25	0.2667	0.2424	0.2540
10 Minute Unigrams 3	12	18	19	0.4000	0.3871	0.3934
10 Minute Unigrams 4	11	19	22	0.3667	0.3333	0.3492
10 Minute Unigrams 5	8	22	24	0.2667	0.2500	0.2581
10 Minute Unigrams 6	9	21	23	0.3000	0.2813	0.2903
Average				**0.3389**	**0.3167**	**0.3274**
10 Minute Bigrams 1	2	21	7	0.0870	0.2222	0.1250
10 Minute Bigrams 2	1	23	8	0.0417	0.1111	0.0606
10 Minute Bigrams 3	2	9	8	0.1818	0.2000	0.1905
10 Minute Bigrams 4	3	9	8	0.2500	0.2727	0.2609
10 Minute Bigrams 5	1	17	8	0.0556	0.1111	0.0741
10 Minute Bigrams 6	2	16	8	0.1111	0.2000	0.1429
Average				**0.1212**	**0.1862**	**0.1468**

Fig. 3: Table of precision, recall, and F-measure scores for both unigrams and bigrams from analysis of data sets consisting of six one-hour segments and six ten-minute segments of tweets from the Twitter Streaming API.

as trending topics by the experimental method. Precision is defined as

$$P = \frac{TP}{TP + FP}$$

where TP is the number of true positives and FP is the number of false positives. Recall is defined as

$$R = \frac{TP}{TP + FN}$$

where TP is the number of true positives and FN is the number of false negatives. The F-measure is the harmonic mean of the precision and recall, defined as

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

The second experiment was evaluated using recall and relevancy scores. Recall was calculated in comparison to the trending topics identified by Twitter using two different methods of identifying true positives and false negatives. The first method only identified as true positives terms that exactly matched terms identified by Twitter as trending topics. Since the second experiment returned only bigrams, terms identified by Twitter as trending topics that were not identified by the experimental method were only considered to be false negatives if they were unigrams. The second method identified a term as a true positive if it either exactly matched a term identifed by Twitter as a trending topic or if it matched one part of a multigram trending topic. Any term identified as a trending topic by Twitter that was not identified as a trending topic by the experimental method was classified as a false negative. Relevance was calculated based on the evaluations of human volunteers. Volunteers were given a list of terms identified as trending topics and marked those that they felt were valid or relevant topics. The list contained both terms identified as trending topics by the experimental method and terms identified as trending topics by Twitter as a control. Relevance was calculated in the same manner as precision was calculated in the first experiment.

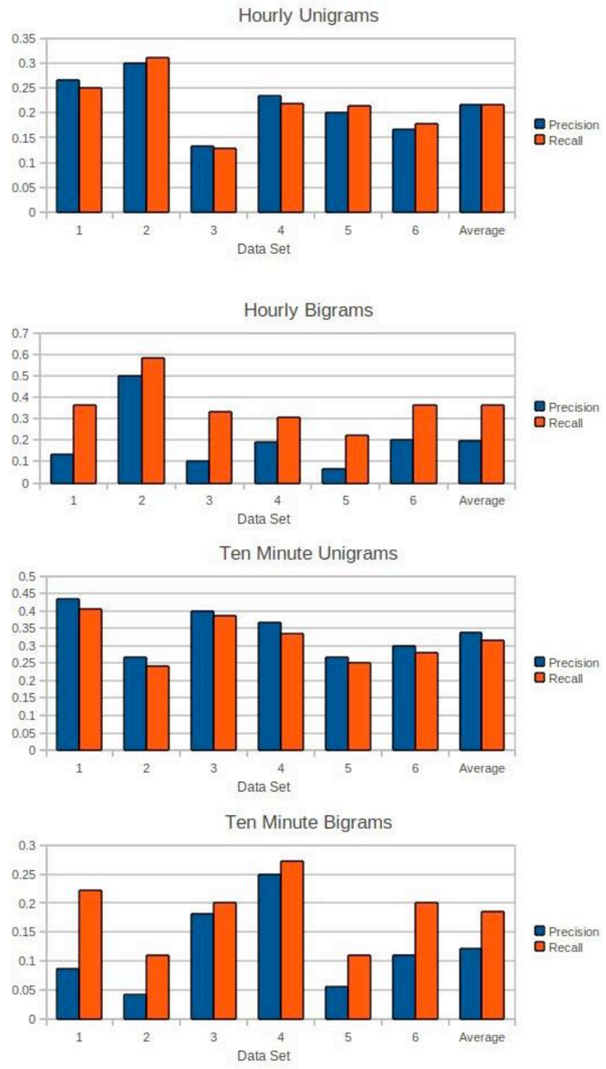
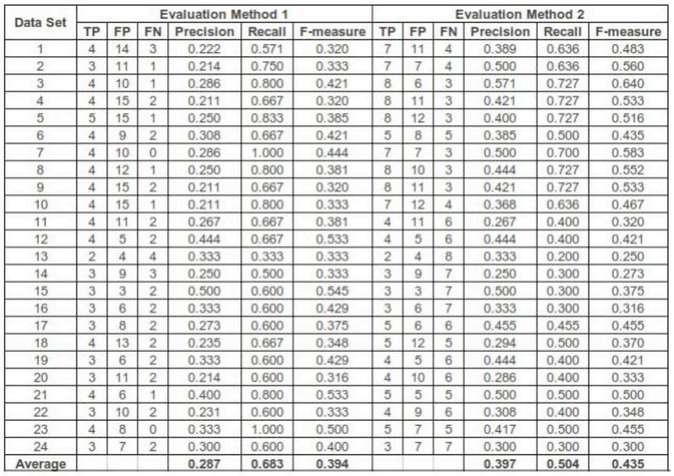


Fig. 4: Graph of precision and recall scores for both unigrams and bigrams from analysis of data sets consisting of six one-hour segments and six ten-minute segments of tweets from the Twitter Streaming API.

VII. Results

For the first experiment, the hourly data sets had an average precision of 0.2167 and 0.1984 and an average recall of 0.2168 and 0.3623 for an F-measure of 0.2167 and 0.2564 for unigrams and bigrams, respectively. The ten minute data sets had an average precision of 0.3389 and 0.1212 and an average recall of 0.3167 and 0.1862 for an F-measure of 0.3274 and 0.1468 for unigrams and bigrams, respectively. A table of the results of the first experiment can be seen in Fig. 3 and a graph showing the precision and recall scores for each data set is shown in Fig. 4. Initial goals for this experiment were a precision score of at least 0.50 and a recall score of at least 0.75, for an f-measure of at least 0.60. The initial results are well below this, but within reasonable range the results of similar work, which produced f-measures in the range of 0.30 to 0.60 [1], [20].

For the second experiment, initial results gave an average precision of 0.2780 and an average recall of 0.7667 for an F-measure of 0.3988 as calculated by the first method of evaluation, and an average precision of 0.4075 and an average recall of 0.5985 for an F-measure of 0.4794 as calculated by the second method of evaluation. The initial results were evaluated by human volunteers as containing relevant topics 72.43% of the time, compared to 77.14% of the time for the terms identified by Twitter as trending topics. Substituting relevance scores for precision scores produces an F-measure of 0.7508 as evaluated by the first method of evaluation and and F-measure of 0.66 as evaluated by the second method of evaluation Given that the success criteria were a recall of 0.50 when evaluated with the terms identified by Twitter and a relevance of at least 75% that of the terms identified by Twitter, the data from the second experiment meets the conditions of success.



Data Set	Evaluation Method 1						Evaluation Method 2					
	TP	FP	FN	Precision	Recall	F-measure	TP	FP	FN	Precision	Recall	F-measure
1	4	14	3	0.222	0.571	0.320	7	11	4	0.389	0.636	0.483
2	3	11	1	0.214	0.750	0.333	7	7	4	0.500	0.636	0.560
3	4	10	1	0.286	0.800	0.421	8	6	3	0.571	0.727	0.640
4	4	15	2	0.211	0.667	0.320	8	11	3	0.421	0.727	0.533
5	5	15	1	0.250	0.833	0.385	8	12	3	0.400	0.727	0.516
6	4	9	2	0.308	0.667	0.421	5	8	5	0.385	0.500	0.435
7	4	10	0	0.286	1.000	0.444	7	7	3	0.500	0.700	0.583
8	4	12	1	0.250	0.800	0.381	8	10	3	0.444	0.727	0.552
9	4	15	2	0.211	0.667	0.320	8	11	3	0.421	0.727	0.533
10	4	15	1	0.211	0.800	0.333	7	12	4	0.368	0.636	0.467
11	4	11	2	0.267	0.667	0.381	4	11	6	0.267	0.400	0.320
12	4	5	2	0.444	0.667	0.533	4	5	6	0.444	0.400	0.421
13	2	4	4	0.333	0.333	0.333	2	4	8	0.333	0.200	0.250
14	3	9	3	0.250	0.500	0.333	3	9	7	0.250	0.300	0.273
15	3	3	2	0.500	0.600	0.545	3	3	7	0.500	0.300	0.375
16	3	6	2	0.333	0.600	0.429	3	6	7	0.333	0.300	0.316
17	3	8	2	0.273	0.600	0.375	5	6	6	0.455	0.455	0.455
18	4	13	2	0.235	0.667	0.348	5	12	5	0.294	0.500	0.370
19	3	6	2	0.333	0.600	0.429	4	5	6	0.444	0.400	0.421
20	3	11	2	0.214	0.600	0.316	4	10	6	0.286	0.400	0.333
21	4	6	1	0.400	0.800	0.533	5	5	5	0.500	0.500	0.500
22	3	10	2	0.231	0.600	0.333	4	9	6	0.308	0.400	0.348
23	4	8	0	0.333	1.000	0.500	5	7	5	0.417	0.500	0.455
24	3	7	2	0.300	0.600	0.400	3	7	7	0.300	0.300	0.300
Average				**0.287**	**0.683**	**0.394**				**0.397**	**0.504**	**0.435**

Fig. 5: Table of precision, recall, and F-measure scores for both unigrams from analysis of data sets consisting of 24 fifteen minute segments of tweets from the Twitter Streaming API.

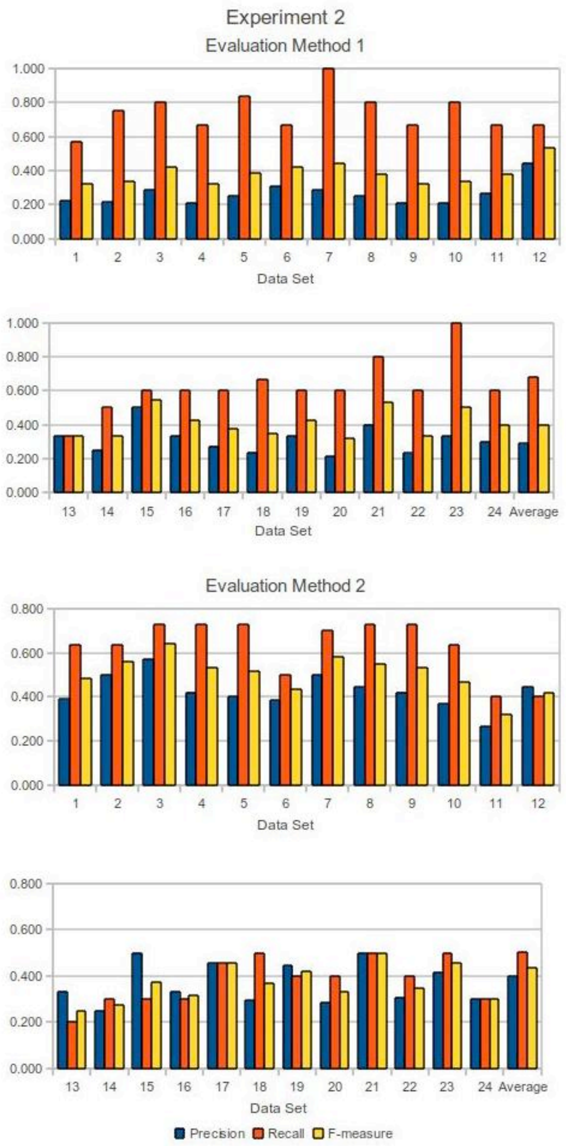


Fig. 6: Graph of precision, recall, and F-measure scores for both unigrams from analysis of data sets consisting of 24 fifteen minute segments of tweets from the Twitter Streaming API.

VIII. Improvements and Extensions

Based on the initial performance of the proposed method, there are several possible extensions and improvements for this project. One potential extention would be to expand the functionality of the unigram and bigram algorithms to identify trigrams or higher order n-grams as trending topics, instead of single words or bigrams. Other possible extensions of this project include interfacing with the Inouye project [6] and the Kaufmann project [7] in order to not only identify but summarize trending topics and normalize the syntax of the summaries, or adapting the method to be used as a predictive tool. One final extension could be in the evaluation process. Terms identified as trending topics could be compared not only to topics identified by Twitter as trending, but to topics identified as trending by other sources, such as Yahoo![8] or Google Trends[9]. Ideally, there will be time for at least two extensions to be implemented during the remainder of the time allotted for this project.

IX. Conclusion

In this paper, we have outlined methodologies for using streaming data, tf-idf term weighting, normalized term frequency analysis, and other criteria to identify trending topics

[8] http://www.yahoo.com

[9] http://www.google.com/trends

on Twitter. The methods implemented detected and identified both unigrams and bigrams as trending topics. Preliminary results for the first experiment fell significantly short of the original goals, but were reasonably close to results produced by other approaches. Preliminary results for the second experiment seem to meet the success conditions put forth in this paper. The current state of the project allows room for extensions in the form of interfacing with other projects applying natural language processing techniques to Twitter. Finally, once all results are analyzed, this project hopefully will have demonstrated the ability of natural language processing tools to extract and identify pertinent information from a continuously changing corpus with an unconventional structure.

References

[1] J. Allan, R. Papka, and V. Lavrenko, "On-line New Event Detection and Tracking," In *Proceedings of ACM SIGR*, pp. 37-45, 1998.

[2] M. Cheong, V. Lee, "Integrating Web-based Intelligence Retrieval and Decision-making from the Twitter Trends Knowledge Base," In *Proceedings of CIKM 2009 Co-Located Workshops: SWSM 2009*, pp. 1-8, 2009.

[3] N. Glance, M. Hurst, and T. Tomokiyo, "Blogpulse: Automated Trend Discovery for Weblogs," In *WWW 2004 Workshop on the Weblogging Ecosystem: Aggregation, Analysis, and Dynamics*, 2004.

[4] D. Gruhl, R. Guha, D, Liben-Nowell, and A. Tomkins, "Information Diffusion Through Blogspace," In *Proceedings of the 13th International Conference on the World Wide Web*, pp.491-501, 2004.

[5] D. Hiemstra, "A probabilistic justification for using tf×idf term weighting in information retrieval," *International Journal on Digital Libraries*, vol. 3, no. 2, pp. 131-139, 2000.

[6] D. Inouye, "Multiple Sentence Microblog Summarization," In *REU Site for Artificial Intelligence, Natural Language Processing and Information Retrieval Research Projects*, 2010. Forthcoming

[7] J. M. Kaufmann, "Syntactic Normalization of Twitter Messages," In *REU Site for Artificial Intelligence, Natural Language Processing and Information Retrieval Research Projects*, 2010. Forthcoming

[8] K. Kireyev, L. Palen, K. Anderson, "Applications of Topics Models to Analysis of Disaster-Related Twitter Data," In *NIPS Workshop on Applications for Topic Models: Text and Beyond*, 2009.

[9] G. Manku and R. Motwani, "Approximate Frequency Counts Over Data Streams," In *Proceedings of the 28th VLDB Conference, Hong Kong, China*, 2002.

[10] R. Nallapati, A. Feng, F. Peng, and J. Allan, "Event Threading within News Topics," In *Proceedings of the Thirteenth ACM Conference on Information and knowledge management,* pp.446-453, 2004.

[11] R. Perera, S. Anand, P. Subbalakshmi, and R. Chandramouli, "Twitter Analytics: Architecture, Tools and Analysis."

[12] S. Petrovic, M. Osborne, and V. Lavrenko, "Streaming First Story Detection with appilcation to Twitter," In *Proceedings of NAACL*, 2010.

[13] S. Petrovic, M. Osborne, and V. Lavrenko, "The Edinburgh Twitter Corpus," In *Proceedings of NAACL Workshop on Social Media*, 2010.

[14] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors," In *WWW2010*, 2010.

[15] G. Salton and C. Buckley, "Term-Weighting Approaches in Automatic Text Retrieval," *Information Processing and Management*, vol. 24, no. 5, pp. 513-523, 1988.

[16] B. Shaparenko, R. Caruana, J. Gehrke, and T. Joachims, "Identifying Temporal Patterns and Key Players in Document Collections," In *Proceedings of the IEEE ICDM Workshop on Temporal Data Mining: Algorithms, Theory, and Applications (TDM-05)*, pp.165-174, 2005.

[17] B. Sharifi, M. Hutton, and J. Kalita, "Experiments in Microblog Summarization," In *NAACL-HLT 2010*, Los Angeles, 2010.

[18] W. J. Wilbur and K. Sirotkin, "The Automatic Identification of Stop Words," *Journal of Information Science*, vol. 18, pp. 45-55, 1991.

[19] W. J. Wilbur and Y. Yang, "An Analysis of Statistical Term Strength and its Use in the Indexing and Retrieval of Molecular Biology Texts," *Computers in Biology and Medicine*,vol. 26, no. 3, pp. 209-22, 1996.

[20] Y. Yang, T. Pierce, J. Corbonell, "A Study on Retrospective and On-Line Event Detection," In *Proceedings of the 21st ACM SIGR*, 1998.

Combining Lexical Resources for Text Analysis of Game Walkthroughs

Michael Billot, University of Colorado at Colorado Springs

Abstract—**One approach to text analysis is motivated by a desire to understand the actions that are most frequent within a body of text. By analyzing words in the text, primarily verbs, connections can be drawn to the actions that are described by the words. The trouble is that single words can have many meanings and evoke many different situations. For that reason, word sense disambiguation software is a vital part of this project. Lexical resources are also needed because they contain two major types of information; the meanings behind words and the relationships between meanings. Another resource needed by this project is a part of speech tagger, which is used for extracting important parts of speech to work with.**

I. Introduction

MANY lexical tools have been developed to assist with computational linguistics and natural language processing. These tools often overlap with their capabilities, but they differ greatly in the ways in which they structure their lexical data. Also, they each have their own strengths and weaknesses. Certain tools often don't cover the same data as other tools. Thus using two tools in conjunction can help populate sparsely covered areas in either or both tools. Using them together may mitigate disadvantages and weaknesses that occur in a single tool. This projects approach to text analysis is based on the assumption that multiple tools used in conjunction are more powerful than any single tool by itself.

II. Motivation

The purpose of this project is to process input text and identify semantic trends, and then use those semantic trends to help generate parameterized actions. Actions are represented by WordNet senses of verbs. An example is the verb open, which occurs fifty-three times in the text. The goal is to output the actions that most frequently occur with the use of the verb open. This is done for every verb that occurs in the text. By doing so, we can produce a resource that dynamically associates words with actions for a specific domain. A possible advantage of this resource is to help disambiguate commands into the actions, such as giving a command to a 3D agent.

III. Related Work

Mihalcea et al. researched various implementations of the PageRank Algorithm for word sense disambiguation, including a combined method that also used the Lesk Algorithm [1].

Banerjee performed word sense disambiguation using WordNet and an adapted Lesk Algorithm. The implementation of his approach comes across as being more simplistic than disambiguation approaches presented in other papers. In the end his disambiguation performance was relatively low in comparison to others [2].

Agirre and Soroa delve into a personalized PageRank algorithm (PPR) that mitigates certain words from having too high of weights [3]].

Giuglea and Moschitti explore ways to connect FrameNet and PropBank via VerbNet [4]. Their process revolves mostly around the parameterized structure of the lexical tools. They define relationships based on common parameterization.

Paziena et al. looked at ways to study verb relations by using WordNet, VerbNet, and PropBank [5]. Their approach was based on the combination of different relational knowledge that each tool provides. WordNet provides verb sense relationships. VerbNet, on the other hand, consists of verb-sense frame knowledge. Their goal was to produce examples of verb pairs that have semantic relations and specific predicate-argument structures.

IV. Approach

This projects approach to performing verb analysis of text is to combine different software and lexical resources. For lexical resources, using their lexical data in combination will strengthen the capabilities of each tool. The other software will allow us to work with the lexical resources in new ways.

A. Domain Used

The selected domain for this project is video game walkthroughs for The Legend of Zelda: Ocarina of Time. The reasoning behind this is that the walkthroughs are freely available online. Also the walkthroughs are full of commands that tell the reader what to do. The action verbs within these commands help the verb sense disambiguation process, whereas sentences with many helping verbs make it more difficult and complicated. Helping verbs are more difficult to disambiguate because their usage in the sentence often does not align well with senses in WordNet.

B. Tools Used

The tools used for this project include the lexical resources WordNet and FrameNet. Another tool is a word sense disambiguator named UKB. Lastly, the Stanford NLP Part of Speech Tagger is used.

1) FrameNet: FrameNet categorizes the English lexicon into semantic frames, which describe situations and the words which compose them [6]. This relationship between a word and a meaning is called a lexical unit. Frames are often associated with multiple lexical units. Within the frame there are frame elements, which are references to supporting frames. Frame elements are divided into two categories: core and non-core frame elements. Core frame elements are mandatory parameters for a frame, and non-core frame elements are optional constraints to the frame. Ultimately, semantic frames correspond to situations, lexical units correspond to the words that evoke those situations (often verbs), and frame elements correspond to the syntactic dependents in a sentence.

2) WordNet: WordNet organizes words into structures called synsets. Synsets encapsulate synonomous words and inter-synset relationships [7]. These synset relationships point to lexical relations of word form, as well as semantic relations of word meaning. Types of relations include hypernymy, hyponymy, antonymy, holonymy, and meronymy.

3) UKB Word Sense Disambiguator: UKB performs word sense disambiguation by using an implementation of the PageRank algorithm [8]. The PageRank algorithm works by ranking nodes, which are normally websites. In this case the nodes are WordNet senses. The reputation of each node is affected by the reputations of nodes that point to it. Therefore, a node with influential nodes pointing to it will have a stronger influence on the nodes that it points to. The following image does a good job illustrating this process.

4) Stanford NLP POS Tagger: The Stanford NLP POS Tagger uses one of two models for English part-of-speech tagging. The first is a model trained on sections 0 through 18 of the Wall Street Journal, uses a left3words architecture, and is 88.85% correct on unknown words. The second model is trained on the same WSJ sections, uses a bidirectional architecture, and can correctly tag 90.46% of unknown words [9]. Either model will be adequate for this project, yet the model using the left3words architecture may be a better option because it takes significantly less time to part of speech tag sentences.

C. Steps

The two primary goals are to sense disambiguate verbs and to align senses with frames. There are a few steps involved to accomplish these two goals.

1) Part of speech tag each sentence in the text.
2) Extract parts of speech from the tagged sentences and store verb frequencies and verb senses in memory
3) Generate context groups for each sentence that contain the verbs, nouns, adjectives, and adverbs of a sentence.
4) Feed the context groups into a word sense disambiguator to get the best verb senses.
5) For the verbs that occur in the text, align their senses with frames in FrameNet.
6) Write the parsed information to a file. This info includes verb frequencies, verb sense IDs, sentences, context groups, and extracted parts of speech.
7) Load the parse information, verb disambiguation results, and frame-sense alignment results into a GUI program to review the results.

1) Word Extraction: After the sentence has been part of speech tagged then the key words must then be extracted from it. This collection of words essentially represents the sentence. There are two types of undesirable collections of words. The first is a sentence that has no verbs identified, either because the part of speech tagger failed or it was never a complete sentence to begin with. This type of sentence is useless because it has no verb to disambiguate. Another undesirable case is where there are less than three parts of speech in the sentence. If there isnt enough contextual information, then verb sense disambiguation cannot perform accurately. Otherwise, if there is enough contextual information, then a context group is made with the sentences parts of speech. The context groups are then passed to the UKB word sense disambiguator.

2) Verb Sense Disambiguation: Word sense disambiguation is performed solely on the verbs of the text. However, the nouns, adjectives, and adverbs will help with this process. UKB will perform verb sense disambiguation using graph-based and lexical similarity methods using a pre-existing WordNet knowledge base [8]. These methods are founded on the PageRank algorithm, variations of which are explored by Agirre and Lopez [10]. Verb sense disambiguation in particular is more difficult and less accurate in comparison to nouns and adjectives [3].

A primary goal of this project is to figure out methods to increase the verb disambiguation performance. An example would be to figure out what is keeping UKB from getting the right answer. After finding what it needs to generate the correct answer, the missing parts can be added to the context group. This process is sort of like nudging UKB in the right direction. Also, one could ask the reverse; what is leading it to the wrong answer? There are sometimes words or word relations that cause UKB to favor one sense. If these could be identified, then they could be manipulated or removed. Lets face it, WordNet has a very high level of granularity. This causes it to have a surplus of word-sense relationships, some of which contribute to improper sense disambiguation.

3) Frame-Sense Alignment: Frame-sense alignment involves aligning verb senses with the FrameNet frames that describe them. This process is not dependent on the results of verb sense disambiguation, since frame alignment works with all senses individually.

A method inspired by Burchardts frame assignment system will be used. As Burchardt suggests, frame assignment may not just involve a single word, but synonyms and hypernyms of that word [11]. The first step involves generating a set of WordNet relatives of the target sense, which includes synonyms, hypernyms, and antonyms. Each relative word then helps score frames. If a relative word evokes a frame, then that frame has the following added to its score: $(framesEvoked(relativeWord))^{-1}$.

If a WordNet relative word evokes many frames, then it has a higher probability of ambiguity. This is taken into account by dividing by the number of frames evoked. Doing so renders ambiguous words to have less of an impact on the scoring of frames. Therefore, relative words that evoke fewer frames are more valuable in the scoring process.

To select the best frame, each frames scores are summed up and the frame with the highest score is the winner. Other methods that measure lemma relatedness to a frame are experimented by Nuges and Johansson [12].

For certain verb senses there are not enough relative words to deem the best frame. If there is no winning frame, then the sense will be considered to be frameless. Another unique case is where two or more frames tie. In this case all the frames will be considered to be the best frames. Senses that perform poorly in frame-sense alignment are in general less common in English.

D. *Output*

The final output will show semantic information about the input text. Most of the information will regard the verbs within the text. Here is a list of the primary displayable data.

- Frequencies of verbs
- Frequencies of disambiguated verb-senses
- Frequencies of actual verb-senses
- Accuracy of verb sense disambiguation
- Best fit frames for each sense
- Pronunciation of words using dictionary.com

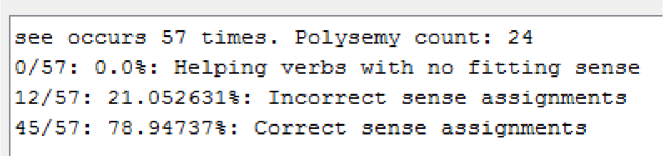
The processes of disambiguating verbs and aligning senses to frames is very time consuming, especially for verbs that have a high number of relationships in WordNet. For that reason the project consists of two programs. The first is a program that generates all the information, and the second is a GUI program that displays it.

E. *Metrics*

There will be two main metrics used to assess the success of the project. The first of which will examine the accuracy of the verb-sense disambiguation. The second will examine the accuracy of the frame-sense alignment.

1) Sense Disambiguation Metrics: In order to determine the disambiguation accuracy one must manually tag each instance of a verb with its real sense, if any. A disadvantage to accuracy occurs because usually a verb is sense tagged by a single person. A remedy would be to have multiple people disambiguate the same sentence in order to assure a more accurate metric. However, it is costly enough to sense tag each verb by hand, so at this point each verb is sense tagged by one person.

After all of the instances of a verb have been disambiguated both by UKB and by hand, then we have the sense disambiguation accuracy. The accuracy information includes the percentage helping verbs with no fitting sense, and the percentage of correct and incorrect disambiguated senses. The following image is an example showing the results for the verb see.



```
see occurs 57 times. Polysemy count: 24
0/57: 0.0%: Helping verbs with no fitting sense
12/57: 21.052631%: Incorrect sense assignments
45/57: 78.94737%: Correct sense assignments
```

2) Frame Alignment Metrics: Frame alignment accuracy is determined manually. If the best fit frame accurately matches the action going on in the sentence, then it is deemed to be correct. A better approach would be to use an existing mapping between WordNet and FrameNet. Unfortunately, there are no available mappings between their current versions, WordNet 3.0 and FrameNet 1.2.

The frame alignment metric provides an accuracy measurement for each verb. It is designed to treat the more prominent senses of a verb as being more important. The first sense of a verb is always the most frequent according to sense annotated data. Therefore, its correct frame alignment should be more important than correctly aligning an uncommon sense.

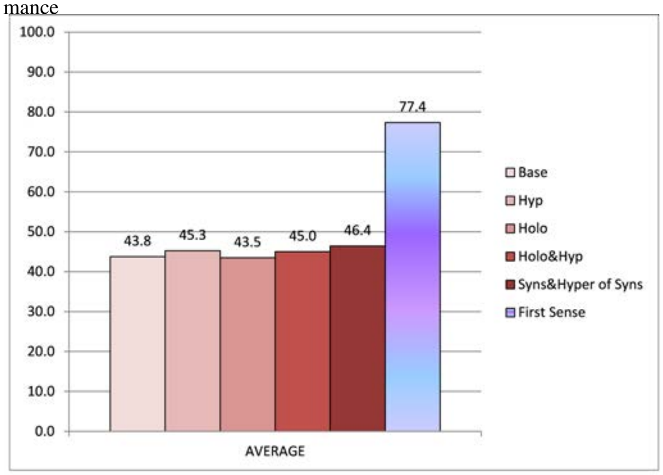
Say a verb has a polysemy count of n. Its highest possible score is $0.5\left(n^2+n\right)$, which is equivalent to $\sum_{i=1}^{n} i$. If the first and most common sense is aligned to the correct frame, then it receives n points. If the second is correct, then it receives n-1 points. If the least common sense is correct, it receives 1 point. The points a verb scores divided by its highest possible score gives its sense-frame alignment accuracy.

V. Results

The results to the projects experiments fall into two separate areas. The first is sense disambiguation results and the other is frame-sense alignment results.

A. *Sense Disambiguation Results*

Fig. 1. Average Personalized PageRank Verb-Sense Disambiguation Performance



Sense disambiguation results are based off of an analysis of the top ten most frequently occurring verbs. These verbs are use, kill, hit, see, open, shoot, walk, find, and fall. There are some verbs that are frequent but excluded from the results. One such verb is be, which is used as a helping verb extensively. Other common verbs that are frequently used as helping verbs frequently include get, go, and do. When verbs are used as helping verbs they are usually difficult to disambiguate because they dont always have a fitting WordNet sense.

There are disambiguation results for six different methods. Five of them use the Personalized PageRank algorithm supplied by the UKB software. They differ in the way that they construct context groups. The base method simply disambiguates the verbs based on the verbs, nouns, adjectives, and adverbs that are in the same sentence. The other methods include WordNet relatives of the nouns in the context group. These relatives are hypernyms, holonyms, and synonyms. A final disambiguation method is called the first sense method. It always assumes a verb is being used in its first sense.

There is a reason for using WordNet relatives of nouns. Nouns are sometimes too vague or unrelated to the verb. The hypernyms and holonyms of nouns could make the context groups more descriptive. For example, in the text there are many instances of the verb kill. The text usually talks about killing monsters in game. Adding in hypernyms of what is being killed may help the Personalized PageRank algorithm develop a stronger connection to a specific sense of kill.

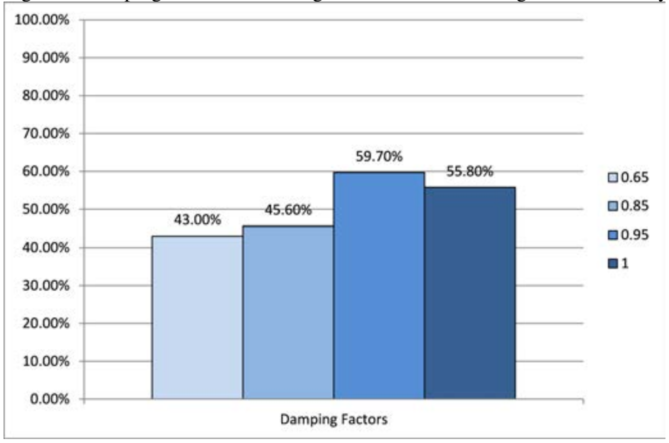
There are also disambiguation results from testing different damping factors. The default damping value is 0.85, which is the recommended value. However, changing the damping value does produce some effects. The lower the damping factor then the faster the iterations will converge on a sense[13]. With a high damping factor, nodes will increase their page rank more quickly. It will be interesting to see how the damping factor influences sense disambiguation.

1) 5.1.1 Average VSD Performance of Different Methods: The following graph shows the average sense disambiguation accuracy of the six different methods. The methods are the following:
1) Base: Context group consists only of verbs, nouns, adjectives, and adverbs.
2) Hyp: Hypernyms of nouns are added to the context group.
3) Holo: Holonyms of nouns are added.
4) Hyp&Holo: Holonyms and hypernyms of nouns are added.
5) Syns&Hyper of Syns: Synonyms and hypernyms of synonyms are added.
6) First Sense: The verbs first sense is always chosen.

Figure 1 shows that verb disambiguation performance does not change drastically when adding WordNet relatives to the context groups. In comparisson to the base results, the addition of WordNet relatives created a 2.6% performance increase at the most. The best performing methods, albeit a small performance advantage, were the ones that included hypernyms. On average, adding hypernyms caused a 1.8% performance increase. Future tests will be done to test other combinations of WordNet relatives. One particular test is adding in WordNet relatives of the verb itself. Potentially, a certain combination of WordNet relatives could cause a performance increase a few percent higher.

One method with a staggering success rate is the first sense method. This method performs so well because the first sense is usually the most common by far. On average it performed 72.6% better than all the other methods. For this reason, it is likely that a good apporach to verb sense disambiguation may involve weighing the first sense of a verb more heavily than the other senses.

Fig. 2. Damping Factor vs. Average Verb-Sense Disambiguation Accuracy



2) Damping Factor : Three different damping factors were tested in addition to the default damping factor of 0.85. The other tested values were 0.65, 0.95, and 1.0. The graph below shows the average sense disambiguation performance of the ten frequently occurring verbs.

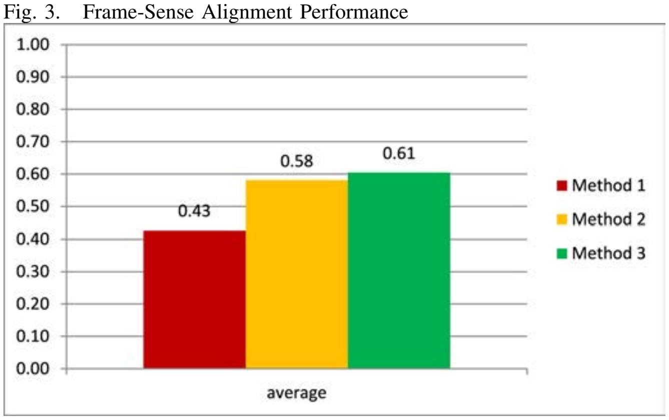
It appears that the default damping factor for the personalized pagerank algorithm may not be the best option for verb disambiguation accuracy. Not even the documentation for the software recommends using a different damping factor. The documentation only mentions that the default damping factor is 0.85. However, based of of the 14.1% increase between the 0.85 and 0.95, one can conclude that there is an optimum damping factor.

The most accurate damping factor is not necessarily the highest possible damping factor. The damping factor of 1.0 actually performed less accurately than the factor of 0.95. In the future it would be nice to test a range of damping factors with small increments of 0.01 or 0.02. That way an optimum damping factor for a body of text could be identified. Other text documents could be analyzed and their optimum damping factors could be compared. Judging by the results expressed in figure 2, the optimum damping factor is probably somewhere around 0.95.

B. Frame Alignment Results

There are three different methods for performing frame-sense alignment. All the methods are based off of using different WordNet relatives of the senses. The three different methods use the following WordNet relatives to score the best frame. Note: frame-sense alignment will not be continued in

Fig. 3. Frame-Sense Alignment Performance



future research. See the future research section for details.

Method 1: Synonyms, hypernyms
Method 2: Synonyms, hypernyms, hyponyms, antonyms
Method 3: Synonyms, hypernyms, hyponyms, antonyms, holonyms, meronyms

The difference in accuracy between method two and method one clearly shows an increase in frame-sense alignment performance because of the additon of hyponyms and antonyms. When comparing methods two and three, the results do not demonstrate a significant impact because of the addition of holonyms and merynyms. Upon further investigation, it looks like verbs usually have no holonymy or meronymy relationships.

VI. Future Work

In future work, there will be some drastic changes to this project. First of all, sense-tagged text will be used. In the event that there isn't enough sense tagged text to produce definitive results, then we may have to resort back to hand tagging verb senses. There are a couple of sense-tagged resources that could be good candidates for future resources. The first is sense-tagged text from the Senseval3 competition [13]. One potential problem is that this data is tagged using WordNet 1.7 senses, and the most current version of WordNet is 3.0. A mapping between WordNet 1.7 and 3.0 would have to be used in order to effectively use this resource. Senseval is currently in its firth competition. The fourth and fifth competitions might also provide sense-tagged text. In end, if there is not enough sense-tagged data to work with, then hand tagging will have to be done.

Another change to the project is the exclusion of FrameNet and frame-sense alignment. Frame-sense alignment was originally included to assist with the long term goal of parameterized action representation. However, that goal is unachievable right now because of how inaccurate verb sense disambigation is. For that reason, the sole approach in the future will be to improve verb sense disambiguation performance.

Future work will involve verb sense disambiguation using the Personalized PageRank method provided by UKB. The goal is to optimize Personalized PageRank's as much as possible. UKB also provides other word sense disambiguation methods besides Personalized PageRank. Even though their optimization is not a focus, their performance could also be observed by applying the same changes made to Personalized PageRank.

There are a few general ways to improve performance of PPR. The first is to manipulate the input given to PPR, which means changing the contextual information given to PPR. Another way to improve performance is to adjust the options of PPR, such as the damping factor and stopping threshold. A final approach is to adjust the source code itself, such as changing the way PPR applies initial weights.

For the manipulation of input, future tests will closely resemble the previous ones. However, they will be more thorough because they will incorporate tagged data. The manipulation of input will still involve adding in WordNet relatives. Other methods could be devised which remove contextual words that are too distant from the target verb. A more efficient metric will be developed to analyze the results quickly so more tests can be done, in comparison to the mere five tests already done. The adjustments of PPR options will definitely look more deeply into using different damping factors. Other options that will be explored are different numbers of PageRank iterations, and different stopping thresholds. Finally, adjustments to the source code could involve any number of things. Currently I would like to focus on having the program apply a higher weight to the first sense of a verb. High weights could even be applied to the first couple of sense for a verb, because the first few senses are always the most common.

Once the adjustments are made and tested, then the best adjustments can be selected and combined. The best performing contextual input format, damping factor, stopping threshold, and first sense weighs will be combined together to achieve higher verb sense disambiguation performance. This approach is critically dependent on the higher weights of first senses. If that implementation is successful, then performance rates could be above 80%, considering that the first sense method performed at about 77%. PPR performance above 80% would be exciting because PPR baseline performance was only 43.8% for this project. Agirre and Soroa's PPR performance using WordNet 3.0 lexical knowledge base was only 41.5%.

References

[1] R. Mihalcea, P. Tarau, and E. Figa, "Pagerank on semantic networks, with application to word sense disambiguation," in *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*. Morristown, NJ, USA: Association for Computational Linguistics, 2004, p. 1126.

[2] S. Banerjee and T. Pedersen, "An adapted lesk algorithm for word sense disambiguation using wordnet," in *CICLing '02: Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*. London, UK: Springer-Verlag, 2002, pp. 136–145.

[3] E. Agirre and A. Soroa, "Personalizing pagerank for word sense disambiguation," in *EACL '09: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. Morristown, NJ, USA: Association for Computational Linguistics, 2009, pp. 33–41.

[4] A. Giuglea and A. Moschitti, "Knowledge discovering using framenet, verbnet and propbank," p. 6, 2004.

[5] M. T. Pazienza, M. Pennacchiotti, F. M. Zanzotto, and V. B. Arcimboldi, "Mixing wordnet, verbnet and propbank for studying verb relations."

[6] [Online]. Available: http://framenet.icsi.berkeley.edu

[7] [Online]. Available: http://wordnet.princeton.edu/wordnet

[8] [Online]. Available: http://ixa2.si.ehu.es/ukb

[9] [Online]. Available: http://nlp.stanford.edu/software/tagger.shtml

[10] E. Agirre, O. L. D. Lacalle, and A. Soroa, "Knowledge-based wsd on specific domains: Performing better than generic supervised wsd."

[11] A. Burchardt, K. Erk, A. Frank, A. Burchardt, K. Erk, and A. Frank, "Lecture a wordnet detour to framenet."

[12] R. Johansson and P. Nugues, "Using wordnet to extend framenet coverage," *In Proceedings of the Workshop on Building Frame-semantic Resources for Scandinavian and Baltic Languages, at NODALIDA, Tartu, Estonia, May 24*, 2007.

[13] [Online]. Available: http://www.senseval.org

Aligning Wiktionary With Natural Language Processing Resources

Ben Casses
Western Carolina University
Cullowhee, NC 28723
bncasses1@catamount.wcu.edu

Abstract

Recently, significant progress has been made towards mapping various natural language processing resources together in order to form more robust tools. While most efforts have gone towards connecting existing tools to each other, recently several projects have involved aligning the popular NLP resources to open collaborative projects such as Wikipedia. Such alignments are promising because they link the specific but frequently narrow NLP data to high coverage open resources. This project explores the effectiveness of some variations of the Lesk Algorithm in connecting specific Wikipedia senses to corresponding senses in other NLP resources. The purpose of this project is to present a potential method of semiautonomous alignment for Wiktionary that will serve to augment other NLP resources.

1 Introduction

Existing natural language processing (NLP) resources can be used in each step of the language comprehension task. Some resources also contain mappings to others. Since some extensive tasks like language comprehension involve the use of multiple resources, such mappings can be very useful. This project will study some methods that may help to reinforce or partially automate mappings between some of these resources through the use of domain based disambiguation and gloss comparisons as in the Lesk Algorithm (Lesk, 1986). The following introduces some of the popular NLP resources and discusses their relative advantages.

1.1 FrameNet

FrameNet is a collection of semantic frames. It contains detailed breakdowns of the function and contextual meaning of a given verb along with all possible participating semantic participants and examples. FrameNet is organized into a hypernym-hyponym hirearchy with more specific terms inheriting structure from their hypernyms. FrameNet's exhaustive detail into semantic participants makes it valuable for studying distance relationships between frames. *Robbery*, for example, inherits directly from *Committing_crime*, indirectly from *Misdeed* and uses *Theft*. [1]

1.2 OntoNotes

OntoNotes is a collaborative project that aims to produce "richer model of text meaning" (Hovy et al., 2006). OntoNotes contains 2,445 verb entries organized into different senses with brief definitions and examples. OntoNotes is the most externally connected of the sense based NLP resources, it contains mappings to FrameNet, PropBank, VerbNet and WordNet. [2]

[1] http://framenet.icsi.berkeley.edu/

[2] http://verbs.colorado.edu/html_groupings/

1.3 PropBank

PropBank contains information on 5,384 verbs. Each verb entry is divided into different syntactic structures based on common use. Structures are subdivided into components *referred to as arguments*. Propbank has value as a parsing and a disambiguating resource. A verb in a given sentence could be matched to one syntactic structure by its surrounding arguments. Once matched, the roles of the arguments are disambiguated according to the roleset. PropBank contains mappings to VerbNet. [3]

1.4 VerbNet

VerbNet is an online database of verbs. Verbs are collected into 274 Levin (1993) style verb classes containing relevant semantic and syntactic information. The value of VerbNet lies in its depth of study into its verb members and their relationships. A given verb may be considered synonymous with its fellow members and hyponymous to its class. This can be useful for the purposes of translation and comprehension if a given verb in VerbNet is not understood but one of its fellow members is. VerbNet contains mappings to FrameNet, OntoNotes, PropBank and WordNet.[4]

1.5 Wiktionary

The effectiveness of Wiktionary in NLP tasks has already been established by Zesch and Müller (2008). At the time of this writing, English Wiktionary had "1,813,199 entries with English definitions from over 350 languages" (wik, 2010). The usefulness of Wiktionary is in its open, collaborative nature. Because anyone can contribute to Wiktionary, it is far more encompassing than any project developed by an individual or more structured group could be. It is also current, while other dictionaries must be revised and updated occasionally, Wiktionary entries are constantly being appended as the nature of the language changes. *The author observed the number of entries increasing by 20,000 over a period of three weeks.* Wiktionary's advantage can also be a disadvantage. Its open nature leads to format inconsistencies between definitions that make automated processing difficult. The potential also exists for incorrect or deliberately misleading entries such as those known to have occurred in Wikipedia (Snyder, 2007) (Chesney, 2006).

1.6 WordNet

WordNet is a long term project hosted by Princeton University. Entries in WordNet are organized into parts of speech and distingushed by sense. Wordnet is highly interconnected with each term referring to related terms including hypernyms, hyponyms, troponyms, frames and peers. Because of its interconnectedness, WordNet is useful for sense disambiguation. An unknown term could be generalized to its hypernym, for example: if the term "shuffle" is not understood, its hypernym, "walk" may be. [5]

2 Related Work

Several different mappings between existing NLP resources already exist. Combinations of resources have been created in different ways with most projects involving multiple alignment techniques to improve accuracy.

- Through common fields shared by both tools. (Loper et al., 2007) (Pazienza et al., 2006) (Giuglea and Moschitti, 2004)
- Through mutual restrictions, where a given entry in one database could match with a given entry in another database, but restrictions prevent this combination, thus narrowing the possiblilities. (Shi and Mihalcea, 2005) (Loper et al., 2007) (Giuglea and Moschitti, 2006)
- Through frequency analysis such as greatest numbers of common synonyms. (Shi and Mihalcea, 2005) (Giuglea and Moschitti, 2006) (Giuglea and Moschitti, 2004)

[3] http://verbs.colorado.edu/propbank/framesets-english/

[4] http://verbs.colorado.edu/verb-index/index.php

[5] http://wordnetweb.princeton.edu/perl/webwn

- Through supervised learning techniques where connections are trained. (Loper et al., 2007) (Giuglea and Moschitti, 2004)
- And through manual mapping of connections. (Pazienza et al., 2006) (Shi and Mihalcea, 2005).

In general, these methods could be divided into three categories: manual methods that require human control, statistical methods that involve matching around comparisons, and learning methods that involve machine learning techniques.

Zesch and Gurevych (2010) compared the effectiveness of several different semantic relatedness analysis methods. They considered four distinct semantic relatedness measures:

- Path based, where the distance between two terms in a graph is considered *edge counting*.
- Information Content based, where the number of documents containing both terms is considered.
- Gloss based, where the quantity of common words in each term's gloss is considered.
- Vector based, where vectors are constructed from multiple documents and the frequency of occurrence of the given term in each document is considered.

Recently work has begun on mapping and utilizing the collaborative resources such as Wiktionary and Wikipedia. The Ubiquitous Knowledge Processing Lab has developed api's for both Wiktionary[6] and Wikipedia[7] and made use of them as NLP resources (Zesch et al., 2008).

3 Problem Definition

The task of aligning NLP resources involves several issues. Each of the NLP resources considered in the introduction have different strengths, but some might be more difficult to align to Wiktionary or less effective when combined. There are structural concerns where two resources do not follow the same layout or present the same information. There are also potential problems where two resources do not expose the same granularity. Only two resources containing the same information could have a one-to-one cardinality.

3.1 Structure and Organization

Not all of the NLP resources discussed in the introduction follow the same structure. WordNet, for example is categorized by frames where each frame contains multiple syntactic structures with interchangable member verbs. Entries in PropBank, however, are centered around a specific predicate or term and divided into usages.

Wiktionary is organized by term with each term divided into different senses. It will be more meaningful to map the Wiktionary senses to the senses of another NLP resource that is organized similarly. Of the resources organized in this way that were discussed earlier, OntoNotes offers the most advantages due to its interconnectedness. A given Wiktionary sense mapped to OntoNotes could be followed to each other resource that OntoNotes is already connected to.

3.2 Granularity

Term-to-term matching is generally trivial, involving only a mutual lookup for a given term. Sense-to-sense mapping becomes more difficult for several reasons. A sense-to-sense connection between two different resources indicates that both senses could be considered to be the "same", but the two senses will generally not contain the same information. For example, one Wiktionary sense of the verb *make*, "To indicate or suggest to be", was aligned to "cause to become, or to have a certain quality" in OntoNotes even though both senses contain different information. Additionally, because of the different ways these resources were constructed, they feature a different degree of granularity around their terms. *Arrive*, for example, has three senses in Wiktionary and one in Ontonotes. Granularity differeneces indicate that the cardinality of this mapping will be many-to-many.

[6] http://www.ukp.tu-darmstadt.de/research/software/jwktl/

[7] http://www.ukp.tu-darmstadt.de/research/software/jwpl/

3.3 Size

Wiktionary is a rapidly growing resource. With the observation that on Wiktionary there may be up to 1,000 new definitions added and many existing definitions modified daily, manual mapping is not a feasable method of alignment. Results would quickly become incomplete or inaccurate. Since it is universally editable, it is not guaranteed that all Wiktionary entries follow the same layout. In some cases formats are inconsistent, information may also be missing, incorrect or presented out of order. Inconstistency makes automated retrieval and matching difficult.

4 Proposed Solution

Due to the differences in content and the lack of existing connections, a gloss based method was selected for aligning Wiktionary to OntoNotes. One advantage of using a gloss method for comparison is that it can act in a naïve fashion. No sense content need be understood or categorized, a sense is just a “bag of words”. Even in occasions where sense parsing does not work as anticipated due to format inconsistency, for example, gloss comparisons will not suffer significantly. A missed tag from one resource, such as *</title>*, is highly unlikely to have a correspondance in another resource.

The gloss comparison in this project will consider common *n*-grams between the compared documents or senses as an indication of similarity. Consider the existence of the uncommon unigram *lathe* in table 1.

Wiktionary
turn: To shape (something) symmetrically by rotating it against a stationary cutting tool, as on a **lathe**.
OntoNotes
Shape by rotating, Examples: After purchasing the wood, I ripped all the pieces to length, then turned the legs on a motorized **lathe**.

Table 1: aligned Wiktionary [10] and OntoNotes [11] senses of turn

Lathe does not appear in any other senses from either resource therefore it indicates a relationship between these two senses.

5 Experiments

Two different experiments were performed, the first involved comparing Wiktionary senses directly to OntoNotes senses. Both experiments attempted to determine the most appropriate sense-to-sense mapping for fifteen verbs, show in table 2, that were selected from three sets of driving directions from Google Maps[12], Yahoo Maps[13] and MapQuest[14].

For each comparison between two documents, or one document and a corpus, a similarity value was computed as the sum of the values of each instance of each n-gram in common between the two sources. The comparison with the greatest similarity value was considered to be the correct choice. Ties involving a correct answer were considered incorrect as they did not effectively disambiguate.

5.1 Direct Method

Let N denote the set of senses $\{N_1, N_2, N_3, ..., N_j\}$ for a given term in OntoNotes and let W denote the set of senses $\{W_1, W_2, W_3, ..., W_k\}$ for the same term in Wiktionary. The matching senses are those with the greatest similarity. To compute similarity, let each sense S_a from W and N contain a set of n-grams $\{S_{a1}, S_{a2}, S_{a3}, ..., S_{ap}\}$. The similarity between two glosses is equal to the sum of the values of the intersection of their terms. $Sim_{N_x,W_y} = \sum_{r \in I} V(r)$ where $I = N_x \cap W_y$ and V is a value filter discussed in 5.4. This process derived a closest match $x \in N$ for each sense $y \in W$.

5.2 Transitive Method

The second method, the transitive comparison, involved first comparing glosses for a given term from Wiktionary, W and OntoNotes, N to a set

[12] http://maps.google.com/

[13] http://maps.yahoo.com/

[14] http://www.mapquest.com/

of n-grams in a domain specific corpus C. As with the previous method, similarity scores were calculated based on the values of the intersection of terms, but in this case, a best similarity score was derived separately for OntoNotes and Wiktionary, $Sim_{N_x} = \sum_{r \in I} V(r)$ where $I = N_x \cap C$ and $Sim_{W_y} = \sum_{r \in I} V(r)$ where $I = W_y \cap C$. The senses N_x and W_y with the greatest similarity scores to C were considered the correct sense for the domain and matched to each other. This process derived a single closest disambiguated sense pair $(y \in W, x \in N)$ for each term.

arrive	avoid	bear
become	continue	enter
go	head	keep
make	merge	start
take	turn	welcome

Table 2: driving verbs

The Wiktionary source text was downloaded from a publicly available data dump [15]. The OntoNotes source text was gathered from the OntoNotes [16] site.

5.3 Testing

Volunteers were initially asked to select only the most appropriate sense to the driving domain from Wiktionary and OntoNotes for each of the fifteen verbs in table 2. Of the thirty selections, the volunteers were found to be in agreement on a single sense only 58% of the time. Because this amount of disagreement either indicates a many-to-one correspondance or some incorrect selections by the volunteers, it was decided that further disambiguation was necessary for testing. The volunteers were combined into a single group and asked to decide on a most appropriate sense *or more than one in the case of a deadlock.* The volunteers were then asked to select the most appropriate OntoNotes sense for 63 additional senses in Wikipedia from the fifteen verbs.

After the committee decisions, there were only two situations that involved one Wiktionary sense mapping to multiple OntoNotes senses. For analysis in these cases, each of the selected OntoNotes senses was considered to be equally correct, that is, if the sense programmatically determined to be the most approprite matched any of the *correct* senses, it was considered a success.

There was one trivial situation where a term had only one sense. The verb *arrive* had only one OntoNotes sense, making determination trivial and potentially skewing results, therefore there were 72 non-trivial determinations to be made in the direct gloss comparisons and 30 determinations to be made in the transitive comparisons.

5.4 Filters

To avoid false matches from insignificant, common words such as *the, of, is*, a word frequency list was formed from the Wiktionary data dump. First, all tags were removed leaving only text. Next, a list of the frequency of appearance of each individual word was created. This list was used to construct two filters. The first filter, *gentle* assigned a value $v = -Log_2 \frac{f}{F+1}$ to a given term found f times in Wiktionary where the greatest frequency for any word was F. The second filter, *severe*, was created to determine if more restrictive scoring achieved any better results. Values for the second filter were set to $v = -Log_2 \frac{f \cdot 20}{F+1}$ and assigned to 10^{-10} if zero or less.

Two different metrics were explored for evaluating n-grams. The first was the sum of the values of the terms $V = v_1 + v_2 + ... + v_n$, the second was a geometric mean, $V = \sqrt[n]{v_1 \cdot v_2 \cdot ... \cdot v_n}$. In both cases, the filters created greater emphasis on n-grams containing rare words. This allowed the trigram *the cat is*, for example to be slightly more significant than *cat* alone but not as significant as *sneaky orange cat*. Only unigrams, bigrams and trigrams were counted.

[15] http://dumps.wikimedia.org/enwiktionary/latest/

[16] http://verbs.colorado.edu/html_groupings/

5.5 Direct Comparison

The direct comparisons involved evaluating a given sense in Wiktionary against each sense for the same term in OntoNotes. Within Wiktionary, for example, there are 13 senses for the verb *make*. Each of these senses was compared to the 17 senses for *make* in OntoNotes. For each Wiktionary sense, the sense pair with the greatest similarity value was considered to be the correct mapping, so for *make* there were 13 possible correct mappings. These results were compared to the direct comparisons made by the volunteers.

5.6 Transitive Comparison

The transitive comparisons involved evaluating each sense from a given resource against a domain specific corpus for disambiguation assistance. Only the domain relevant sense decided by the volunteers was considered to be the correct answer, so there was only one correct sense for each verb in each resource.

Two corpora were created for testing transitive comparisons. The first was formed from the sources for the 15 verbs, driving directions from Google, MapQuest, and Yahoo Maps. The second corpus was created from Google searches for "driving +turn +go +keep +bear".

6 Results

For each method, the results were considered compared to the volunteer selections. If the sense pair with the greatest similarity score matched the sense pair selected by the volunteers, the matching was considered correct.

6.1 Direct Comparison Results

For the direct comparisons, there were 72 non-trivial matches. Four sets of experiments were performed: Sum n-gram and Geometric Mean n-gram evaluations were performed with the *gentle* and *severe* filters. Although they resulted in slightly different sets of correct answers, neither evaluation or filter method did significantly better. Results are shown in table 3 as accuracy percentages out of 72 possible matches.

For a baseline comparison, a naïve selection was developed that involved matching all Wiktionary senses to the first OntoNotes sense for a given term. The naïve achieved 35% accuracy.

	Sum Eval	Geom Mean
gentle filter	46%	46%
severe filter	47%	46%

Table 3: Direct Comparisons

6.2 Transitive Comparison Results

For the transitive comparison, there were 29 non-trivial matches. Eight comparisons were performed. Sum n-gram and Geometric Mean n-gram evaluations were performed with the *gentle* and *severe* filters in comparisons to both corpora. In the transitive case, the *severe* filter performed slightly better than the *gentle* filter. Results are shown in table 4 as accuracy percentages out of 29 possible matches.

A naïve method, selecting the first sense was used for a baseline comparison to the transitive method. The naïve method achieved 48%.

original corpus	Sum Eval	Geom Mean
gentle filter	48%	48%
severe filter	48%	52%

Google search	Sum Eval	Geom Mean
gentle filter	48%	48%
severe filter	62%	59%

Table 4: Transitive Comparisons

Discarding all but the most effective methods, these results, 47% and 62% can be compared to (Lesk, 1986) 50%-70% and (Banerjee and Pedersen, 2002) 25% for verbs and 74%-78% for (Kilgarriff and Rosenzweig, 2000).

7 Problems

Although each method performed as good as or better than naïve selection, 72 potential matches among 15 verbs may not be enough to make any determination about the validity of the hypothesis. Further tests are needed to reinforce the effectiveness of these methods.

Some OntoNotes senses were troublesome for gloss comparisons. The fifteenth sense of *go*, for example, "miscellaneous idioms..." is a catch-all sense containing more text, therefore more inadvertent matches, than other senses. The seventeenth OntoNotes sense of *make*, "other verb particle constructions", is informative to a human reader, but contains little information for gloss comparisons.

It is likely that a single document may use the same word in two different senses. While this was not the case with the transitive comparison corpora used, it could cause confusion in future experiments.

8 Conclusion

These results are far better than random, 28% for transitive and 46% for direct comparison. The most accurate mapping method found was 62% for the *severe* filter sum evaluation on the Google search corpus. While both the direct and transitive methods achieved results slightly better than their corresponding naïve methods, 14% and 17% better respectively, the results are not strong enough to regard these experiments as effective stand alone semiautonomous alignment methods.

It appears that the neither method of evaluating n-grams performed significantly better at disambiguating. It is possible that false matches, n-grams that correspond to the incorrect sense, were more significant to the outcome than evaluation methods. This could explain why changing the filter caused more of a difference than changing the evaluation method.

9 Future Work

A larger selection of verbs to disambiguate should help to better establish the accuracy of this method. Additionally, a different domain for transitive comparisons would further prove the possibilities of this method.

The value of the most likely sense, whether correct or not, often stood out strongly from the other senses, sometimes differing in value by an order of magnitude. This could indicate that the analysis method may be "fooled" by false matches. If these false matches could be isolated and disregarded somehow, results might improve.

The corpus, in the case of the transitive mapping, also caused some false matches, for example, the phrase "crossing into NEBRASKA" in the corpus and the example *...water turned into ice...* in OntoNotes caused *turn* to match to the *become* sense in one experiment. Issues like these might be resolved by scaling the values of n-grams based on their distance in the document from the term being considered. The text corpus for comparison could be refined to a domain keyword list which would eliminate some extraneous terms.

The *severe* filter, which performed better than the *gentle* filter, was created after the *gentle* filter in response to the determination that insignificant words still had too much influence. Perhaps a stronger filter could produce better results.

There are several possiblities for augmenting this gloss comparison method. Second order comparisons as used by (Banerjee and Pedersen, 2002) might be beneficial. Additionally, it has been suggested that syntactic limitations from a given sense could be considered in disamgibuation.

Verbs containing troublesome senses as mentioned in the Problems section cannot be aligned through gloss comparisons. In future experiments, such verbs should be discarded from the test set.

References

Satanjeev Banerjee and Ted Pedersen. 2002. An adapted lesk algorithm for word sense disambiguation using wordnet. pages 117–171.

Thomas Chesney. 2006. An empirical examination of wikipedias credibility.

Ana-maria Giuglea and Ro Moschitti. 2004. Knowledge discovering using framenet, verbnet and propbank.

Ana-maria Giuglea and Ro Moschitti. 2006. Semantic role labeling via framenet, verbnet and propbank. In *In Proceedings of COLING-ACL*.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% solution. In *NAACL '06: Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers on XX*, pages 57–60, Morristown, NJ, USA. Association for Computational Linguistics.

Adam Kilgarriff and Joseph Rosenzweig. 2000. Framework and results for english senseval.

Michael Lesk. 1986. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *SIGDOC '86: Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26, New York, NY, USA. ACM.

Beth Levin. 1993. *English Verb Classes and Alternations: a preliminary investigation*. University of Chicago Press, Chicago and London.

Edward Loper, Szu ting Yi, and Martha Palmer. 2007. Combining lexical resources: Mapping between propbank and verbnet. In *In Proceedings of the 7th International Workshop on Computational Linguistics*.

Maria Teresa Pazienza, Marco Pennacchiotti, Fabio Massimo Zanzotto, and Via B. Arcimboldi. 2006. Mixing wordnet, verbnet and propbank for studying verb relations.

Lei Shi and Rada Mihalcea. 2005. Putting pieces together: Combining framenet, verbnet and wordnet for robust semantic parsing. In Alexander F. Gelbukh, editor, *CICLing*, volume 3406 of *Lecture Notes in Computer Science*, pages 100–111. Springer.

Johnny Snyder. 2007. Its a wiki-world utilizing wikipedia as an academic reference.

2010. Wiktionary english version main page.

Torsten Zesch and Iryna Gurevych. 2010. Wisdom of crowds versus wisdom of linguists - measuring the semantic relatedness of words. *Natural Language Engineering*, 16(01):25–59.

Torsten Zesch, Christof Mller, and Iryna Gurevych. 2008. Using wiktionary for computing semantic relatedness. In *In Proceedings of AAAI*.

Event and Temporal Information Extraction towards Timelines of Wikipedia Articles

Rachel Chasin
Department of Computer Science
University of Colorado at Colorado Springs
Colorado Springs, Colorado 80918

***Abstract*—This paper explores the task of creating a timeline for historical Wikipedia articles, such as those describing wars, battles, and invasions. It focuses on extracting only the major events from the article, particularly those associated with an absolute date. Existing tools extract all possible events, while we write tools to identify time expressions and anchor them in real time. From this set of all events, we identify the major ones using a classifier. We then place these events on a timeline and label them with a time interval as small as possible. The timeline is integrated into an online user interface that displays named entities for each event and finds its locations on a map; we separately list named entities and locations mentioned in the article but not around any event.**

I. Introduction

EVENT and temporal information extraction from plain text is a crucial task for natural language processing and knowledge management, particularly in the tasks of summarization and question-answering. Topic summarization must pick out the important events in one or more stories to yield the best summary with the least extraneous information. Question-answering tools must be able to answer queries about dates, durations, and even relative times, whether in natural language or with a set query type (“When did the Civil War end?”, “How long was the Battle of Gettysburg?”, “How many years were between the Civil War and World War I?”).

The possible domains for temporal information extraction are numerous and varied; it is especially useful in the news domain and for patient reports in the medical domain. This paper discusses its use in documents that describe historical events. The problem it addresses can be roughly broken down into two large components: extracting only important events, and relating them via the temporal expressions in the document so they can be viewed on a timeline.

II. Related Research

Even setting aside the problem of automating the process, identifying and representing the temporal relations in a document is a daunting task for humans. Several structures have been proposed, from Allen’s 1983 interval notation and 13 relations [1] and variations on it using points instead, to a constraint structure for the medical domain in [2] (patient hospital reports). The most recent way to represent the relations is a set of XML tags called TimeML.

Software has been developed ([3], [4]) that identifies events and time expressions, and then generates relations among them. Events generally consist of most verbs in the document, but also include some nouns. The link generation is done differently depending on the system, but uses a combination of rule-based components and classifiers.

Research on event extraction has often focused on identifying the most important events in a set of news stories ([1], [5]). The advantage of this domain is that important information is usually repeated in many different stories, all of which are being examined. This allows algorithms like TF-IDF to be used. In this Wikipedia project, there is only one document per specific topic, so these algorithms cannot be used. There has also been research into classifying events into specific categories and determining their attributes and argument roles by [6] (submitted to the ACE task in 2005). Another issue addressed by the ACE task is event coreference, determining which descriptions of events in a document refer to the same actual event.

III. Our Approach

The task of making a timeline for a Wikipedia article lends itself well to separation into two subtasks - identifying whether an event is important or not, and putting these events on a timeline. We focused on each of these separately. After the timeline data was generated, we used the existing software Simile[1] to graphically represent it, and worked with a group researching the identification and mapping of geospatial entities to combine the two visualizations.

A. Important Events

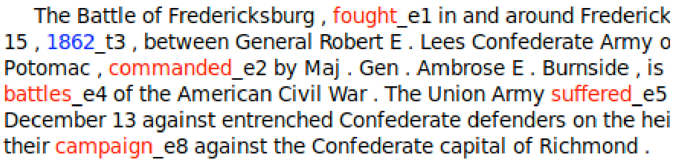
For determining important events, we first run the EVITA program[2] described in [7] on the article, which labels all possible events in the TimeML format. Specifically, it places XML “EVENT” tags around single words defining events; these may be verbs or nouns, as seen in Figures 1a and 1b. Each event has a class; in this case, “fought” is an occurrence. Other classes include states and reporting events. For our purposes, occurrences will be the most important because those are the kinds of events generally shown on timelines.

Rachel Chasin is an undergraduate at Massachusetts Institute of Technology, Cambridge, MA, 02139 e-mail: rchasin@mit.edu.

[1] Simile was originally authored by David Franois Huynh and is available at http://www.simile-widgets.org/timeline/.

[2] The entire TARSQI Toolkit is freely downloadable upon email request; see: http://timeml.org/site/tarsqi/toolkit/download.html

EVITA recognizes events by first preprocessing the document to tag parts of speech and chunk it, then examining each verb, noun, and adjective for linguistic (using rules) or lexical (using statistical methods) properties that indicate it is an event. Instances of events also have more properties that help temporal relations to be established in later processing steps; these properties include tense and sometimes modality.



The Battle of Fredericksburg , fought_e1 in and around Frederick
15 , 1862_t3 , between General Robert E . Lees Confederate Army o
Potomac , commanded_e2 by Maj . Gen . Ambrose E . Burnside , is
battles_e4 of the American Civil War . The Union Army suffered_e5
December 13 against entrenched Confederate defenders on the hei
their campaign_e8 against the Confederate capital of Richmond .

Portion of an article with events tagged by EVITA (words followed by e_i)

```
The Battle of Fredericksburg, <EVENT
class="OCCURRENCE" eid="e1">fought</EVENT>
in and around Fredericksburg, Virginia [...]
```

Figure 1b: A portion of the XML representation of Figure 1a.

For this task, an event is considered to be the sentence containing it. These sentences will eventually be used as the text for each event on the timeline. Each event/sentence is classified as important or not using a classifier trained with a set of mostly word-level and sentence-level features. On the first attempt at classification, one classifier used numerical features and another used purely binary features, translating the former into the latter by putting them in or out of ranges. The numerical classifier used 16 features and the purely binary one used 19. The numerical classifier's features are listed in Table 1 and the purely binary classifier used the same ones with numerical features split into ranges. Many features had to do with the characteristics of the event word in the sentence (part of speech, grammatical aspect, distance from a named entity, etc.) or the word by itself (length). Some had to do with the sentence as a whole (for example, presence of negation and presence of digits).

Two were also related to the article as a whole - position of the sentence in the document, and similarity of the event word to article "keywords." These keywords were taken as the first noun and verb or first two nouns of the first sentence of the article. These were chosen because the first sentence of these historical narratives often sums up the main idea and will often therefore contain important words. In the articles of our corpus, these are often "war," "conflict," or "fought," for example. An event word's similarity to one of these words may having a bearing on its importance. The decision to use two keywords helps in case one of the words is not a good keyword; only two are used because finding similarity to a keyword is expensive in time. Similarity is measured using the "vector pairs" measure from the WordNet::Similarity Perl module, proposed in [8]. This calculates the similarity of the vectors representing the glosses of the words to be compared. This measure was chosen because it was one of the few that can calculate similarity between words from different parts of speech, which was necessary for this feature.

Important Event Classifier Features, version 1
Distance (characters) from nearest named entity
Digit presence in sentence
Negation presence in sentence
Position (by token) of event word in sentence
Position of sentence in article
Length of event word
Similarity of event word to keywords
Event word is capitalized
Event word is noun
Event word is verb
Event word is in past tense
Event word is in infinitive
Event word has no tense (for nouns)
Event word has perfective aspect
Event word has positive polarity
Event word is of the event class ”occurrence”

Table 1: A list of features for classifying events as important or not

Upon training and testing, SVMs using these features performed poorly, particularly in precision (see Experiments), so we altered the approach. Because events are considered to be the sentences containing them, that is, we are trying to decide which sentences are important based on the events EVITA extracts from them, we changed the set of features to apply to sentences. The new set of features discarded some features that did not make sense to apply to the sentence; included the same sentence-level features as well as some new ones; and changed some word-level to sentence-level features by adding, averaging, or taking the maximum of the word-level features for each event in the sentence. The new features can be seen in Table 2.

The most different feature added was a feature based on TextRank ([9]), an algorithm developed by Mihalcea that ranks sentences based on importance and is used in text summarization. TextRank works using the PageRank algorithm developed by Google. While PageRank works on web pages and the links among them, TextRank treats each sentence like a page, creating a weighted, undirected graph whose nodes are the document's sentences. Edge weights between nodes are determined using a function of how similar the sentences are. After the graph is created, PageRank is run on it which ranks the nodes in order of essentially how much weight points at them (taking into account incident edges and the ranks of the nodes on the other sides of these edges). Thus sentences that are in some way most similar to most other sentences get ranked highest. We wrote our own implementation of TextRank with our own function for similarity between two sentences. Our function automatically gave a weight of essentially 0 if either sentence was shorter than a certain threshold (we chose 10 words). For all others, it calculated the “edit distance” between the sentences, treating words (rather than characters) as the units to be compared and calling two words equal if their stems are equal. The similarity was then chosen as the sum of the sentence lengths divided by their edit distance.

Named entities were also considered more important to the process than before. Instead of just asking if the sentence contained one, the some measure of the importance of the named entity is calculated and taken into account for the feature. This is done by counting the number of times the named entity is mentioned (people being equal if their last names are equal, and places being equal if their most specific parts are equal). This total is then normalized for the number of named entities in the article.

Important Event Classifier Features, final version
Presence of an event in the perfective aspect
Percent of events in the sentence with class “occurrence”
Digit presence
Maximum length of any event word in the sentence
Sum of the Named Entity ‘weights’ in the sentence (NE weight being the number of times this NE was number of mentioned in the article divided by the all NE mentions in the article)
Negation presence in the sentence
Number of events in the sentence
Percent of events in the sentence that are verbs
Position of the sentence in the article normalized by the number of sentences in the article
Maximum similarity (as previously described) of any event word in the sentence
Percent of events in the sentence that are in some past tense TextRank rank of the sentence in the article, divided by the number of sentence in the article
Number of “to be” verbs in the sentence

Table 2: A final list of features for classifying events as important or not

B. Temporal Relations

The rest of TTK creates event-event and event-time links (called TLINKs). Times are identified by GUTime, another part of the toolkit, which marks them with “TIMEX3” XML tags. These include attributes like type of expression (DATE, TIME, etc.) and value (when possible to tell). A time identified in Figure 1a was “1862.” Some events, then, will be anchored to time expressions, and some to other events. These are represented by the TLINK XML tag, which has attributes like the type of relation. There are other links, called SLINKs, that describe modal relations between events, but these are not as relevant to the task, though perhaps useful for filtering out events that get tagged but do not actually happen in the narrative (ex. what someone thought would happen). An example of each kind of TLINK is shown in Figure 2. The first represents that fact that the event with ID 1 is related to the time with ID 3 by “before”; the second also represents “before,” between events 5 and 6.

```
<TLINK eventInstanceID="ei1" lid="l9"
origin="CLASSIFIER 0.995194" relType="BEFORE"
relatedToTime="t3"/>
<TLINK eventInstanceID="ei5" lid="l10"
origin="CLASSIFIER 0.999577" relType="BEFORE"
relatedToEventInstance="ei6"/>
```

Figure 2: Two TLINKs.

In fact, almost all the TLINKs we encountered when trying out the TTK linking program were “before” links, and they gave little to no more information reiterating the order of the events in the text. While this is generally accurate, since it can be done without the program, we decided not to use TTK for the temporal relation processing.

As a step before beginning this work, we tried a simple approach just using regular expressions to extract times. The results of using extensive regular expressions versus using the GUTime part of the TTK showed that the regular expressions pull out many more (complete) dates and times. For example, in Figure 1a, GUTime only finds 1862, while the regular expressions would find a range from December 11 1862 to December 15 1862. Because of this, we decided to use our own program based in regular expressions rather than GUTime. A flaw present in our program and not in GUTime is its ability to only pick out one time expression (point or interval) per sentence. This is consistent with our current view of events as sentences, although it would arguably be better to duplicate the sentence’s presence on a timeline while capturing all time expressions present in it. We do not think it would be overly difficult to implement this change although it might cause problems for the extraction of expressions that have parts that can be far away from each other in the sentence.

GUTime also attempts to compute real values for the times in the ISO8601 standard, but usually does it in relation to the document creation time; this is useful for news articles, for which a creation time is provided, but is detrimental in our case, as it assumes the current date. Instead, to anchor time expressions to real times - specifically to a year - we have used a naive algorithm that chooses the previous anchored time’s year. We heuristically choose the most probable year out of the previous anchored time’s year and the two adjacent to it by looking at difference in the month, if it is given. For example, a month that is more than five months earlier than the anchoring’s time’s month is probably in the next year rather than the same year (with “The first attack occurred in December 1941. In February, the country had been invaded,” it is probably February 1942). In addition to the year, if the time needing to be anchored lacks more fields, we fill them with as many corresponding fields from the anchoring time as it has.

Each time expression extracted is considered to have a beginning and an end, at the granularity of days (though we do extract times when they are present). Then the expression “December 7, 1941” would have the same start and end point, while the expression ”December 1941” would be considered to start on December 1 and end on December 31. Similarly, modifiers like “early” and “late” change this interval according to common sense; for example, “early December” corresponds to December 1 to December 9. While these endpoints are irrelevant to a sentence like, “The Japanese planned many invasions in December 1941,” it is necessary to have exact start and end points in order to plot a time on a timeline. Thus while the exact start and end days are often arbitrary for meaning, they are chosen by the extractor.

Some expressions cannot be given a start or end point at all. For example, “The battle began at 6:00 AM” tells us that the start point of the event is 6:00AM but says nothing about the end point. Any expression like this takes on the start or end point of the interval for the entire event the article describes (for example, the Gulf War). This interval is found by choosing the first beginning time point and first ending time point that are anchored directly from the text, and is logically probable to find the correct span. While this method is reasonable for some expressions, many of them have an implicit end point somewhere else in the text, rather than stretching until the end of the article’s event. It would likely be better to instead follow the method we use for giving times to sentences with no times at all, described below.

After initial extraction of time expressions and finding a tentative article span, we discard times that are far off from either end point, currently using 100 years as a cutoff margin. This helps avoid times that are obviously irrelevant to the event, as well as expressions that are not actually times but look like they could be (for example, “The bill was voted down 166-269” which looks like a year range). We also discard expressions with an earlier end date than start date, which helps avoid the latter problem.

Despite the thoroughness of the patterns we look for in the text, the majority of sentences still have no times at all. However, they may still be deemed important by the other part of our work, so must be able to be displayed on a timeline. Here we exploit the characteristic of historical descriptions that events are generally mentioned in the order in which they occurred. For a sentence with no explicit time expression, the closest (text-wise) sentence on either side that does have a time expression is found, and the start times of those sentences are used as the start and end time of the unknown sentence. There are often many unknown sentences in a row; each one’s position in this sequence is kept track of so that they can be plotted in this order, despite having no specific date information past the interval, which is the same for them all. Sometimes the start and end time we get in this manner are invalid because the end time is earlier than the start time. In this case, we look for the next possible end time (the start time of the next closest sentence with a time expression). If nothing can be found, we use a default end point. This default end point is chosen as the last time in the middle N of the sorted times, where N is some fraction specified in the program (currently chosen as 1/3). We do this because times tend to be sparse around the ends of the list of sorted times, since there are often just a few mentions of causes or effects of the article’s topic.

IV. Experiments

A. Important Events

Given tagged data, the “important event extraction” was simple to test since it is a classification problem. It required a set of Wikipedia articles that were hand-tagged as to whether each event word identified by EVITA was part of an important event. An event word being part of an important event meant that the sentence containing it was important and that it contributed to this importance. Because there is no good objective measure of importance (except, perhaps, presence on a manually generated timeline from some other source), we asked multiple volunteers to tag the articles to avoid bias. The subjectivity was a real problem, however. We gave the annotators guidelines for what events were and were not important, but there was still a lot of confusion and disagreement. The most basic guideline was to judge whether you would want the event on a timeline of the article. Other guidelines included to not tag it if it was more of a state than an action, and to tag all event words that referred to the same event. Each article was annotated by more than one person so that disagreements could be resolved using a majority vote. There were 13 articles annotated in total and the breakdown of numbers of annotators was: 1 article annotated by 5 people, 4 articles annotated by 4 people, 6 articles annotated by 3 people, and 2 articles annotated by 2 people.

Originally, different words tagged as events by EVITA were considered different events even if they were in the same sentence. During this stage, an event was labeled important for the classifier if at least half of its annotators tagged it as such. Inter-annotator agreement was calculated pair-wise and averaged over all pairs of annotators of an article. One annotator’s tagging was taken as ground truth, and each other annotator’s accuracy measures were calculated against that. Over all articles, all “ground truth” annotators for that article, and all annotators compared against them, the average F1-score was 42.6%, indicating great disagreement.

An SVM using a radial basis function kernel was trained on 11 articles, with 2 left for testing. This yielded precision, recall, and f-measure scores. The 11 training articles consisted of 5602 event words and the 2 testing articles consisted of 3227 event words (1503 and 1724). The SVM was trained with the LIBSVM software ([10]) using a binary classifier with a radial basis function kernel. The parameters for the kernel and cost constant were found with one of LIBSVM’s built-in tools.

After the initial training and testing, the SVM simply performed as a majority-class classifier, since the number of negative examples - events tagged as unimportant - grossly outnumber the positive examples. This obviously yielded high accuracy and high precision, but almost 0 recall.

Two common methods for combatting sample inequality of classes were considered and tried. One was SMOTE ([11]), Synthetic Minority Over-sampling Technique, presented by Chalwa in 2002. This technique reads in positive examples and creates artificial positive examples close to the real ones. This is better than simple oversampling because it creates different points rather than repeating the same points (although it does end up doing some repetition). We used SMOTE to generate enough new positive examples so that the numbers of positive and negative examples were approximately equal for each article. The other method to prevent simple majority-class classification was weighting the costs of misclassifying positive and negative examples differently. By penalizing misclassified positive examples more than negative ones, the SVM gets trained to make fewer of those errors, increasing recall at the expense of precision. Following [12], we tried both of these methods together, but found that oversampling with SMOTE in addition to having different costs ended up with slightly lower scores. Further, for the numerical feature SVM (as opposed to the pure binary feature SVM), the best cost ratio with oversampled data was just 1. The precision-recall data points for different conditions are given in Figure 3. The different data sets represent the two test articles, each compared with predictions from an SVM that was or was not ("unsmoted") trained on oversampled data.

The results of testing the modified SVM showed low precision and medium to high recall. For the SVM with binary features, the best case was with data that was not oversampled, using a positive example cost to negative example cost ratio of 3. The average F-score between the two testing articles was 33.4% in this case, with precisions of 22.0% and 25.8%, and recalls of 54.0% and 56.8%. The SVM with numeric features performs similarly, with the highest average F-score being 33.3% on data that was not oversampled with a cost ratio of 2. The precisions for this are 29.0% and 27.8%, and the recalls are 40.9% and 39.5%.

We thought these low scores partially stemmed from the fact that annotators may have considered the same sentences important but tagged different event words within them, thus not giving accurate test data. Since we had told them to decide whether sentences were important or not and we were planning to only display sentences on the article timeline, we decided to classify at sentence level as well.

The annotated data was again processed to give a test set of important and unimportant sentences. A sentence was labeled important for the classifier if either (1) some event in the sentence was marked important by every annotator, or (2) at least half of the events in the sentence were marked important by at least half of the annotators. Using this criteria, over the 13 articles 696 sentences were labeled important and 1823 unimportant. The pairwise inter-annotator agreement was recalculated for this new method of labeling and the average F-score between annotators rose to 66.8% (lowest being 47.7% and highest being 78.5%).

As described above, a different set of features was chosen and calculated for each of the 13 articles. SVMs were trained and tested using 10-fold cross-validation. Rather than breaking up the data into training and test sets by article, this time the 2519 sentences were divided into 10 equal sets plus one remainder set that was not used in training or testing during cross-validation.

To determine the best SVM model, the parameter space was partially searched for cost of missing examples (c), the parameter gamma of the radial basis kernel function (g), and the weight ratio of cost for missing positive examples to cost for missing negative examples (w). For any given choice of c, g, and w, an SVM was trained on each cross-validation training set and tested on the corresponding test set, using libsvm. The resulting F-scores were averaged over the different sets and recorded. The search space was explored starting at c=1.0, g=1.0, w=1. c and g went down by powers of 2 and w was incremented. This portion of the space was explored because in the original classification SVM training, the optimal c and g (as found by a libsvm tool) were usually 0.5 and 0.5. Fixing c and w, the next g would be chosen until F-scores no longer went up. w was usually set to 2 almost immediately because of low scores for w=1. This was repeated for w up to 3 (as 4 produced slightly poorer results), and then the next value of c was chosen. The results of these tests are shown in Table 3. 0.5 and 0.5 were optimal values for the event-word-level classifiers, however, so a broader search may have been beneficial.

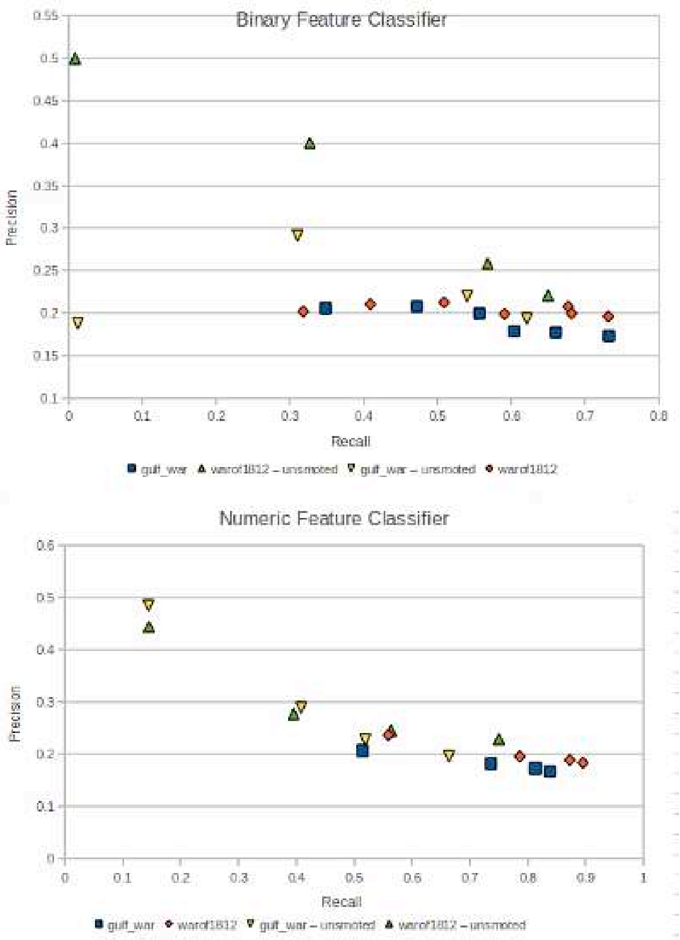


Figure 3: Precision and recall for original SVM results

The best results were at 51.0% F-score and came from a few different choices of parameters. The set c=1.0, g=0.125, w=3 was chosen and the final model was trained with those parameters on the entire data set.

c	g	w	F-score
1.0	1.0	1	29.0
1.0	1.0	2	46.1
1.0	0.5	2	46.9
1.0	0.25	2	47.5
1.0	0.125	2	48.0
1.0	0.0625	2	49.0
1.0	0.03125	2	48.0
1.0	1.0	3	46.2
1.0	0.5	3	48.9
1.0	0.25	3	50.1
1.0	0.125	3	51.0
(p:41.06066, r:75.0522, f:51.0428)			
1.0	0.0625	3	50.9
0.5	1.0	2	45.2
0.5	0.5	2	47.1
0.5	0.25	2	47.5
0.5	0.125	2	48.2
0.5	0.0625	2	48.1
0.5	1.0	3	48.0
0.5	0.5	3	49.2
0.5	0.25	3	50.55
0.5	0.125	3	51.0
(p:41.14866, r:74.7939, f:51.03596)			
0.5	0.0625	3	51.0
(p:41.35864, r:73.9689, f:51.01095)			

Table 3: SVM parameter searching test results (averaged over cross-validation sets)

B. Temporal Relations

Testing the times is significantly more difficult, since the intervals generated by this program are certainly different than the ones intended by the article representing events with times is difficult even for human annotators. Instead, we will use a measure proposed by Ling and Weld in 2010 ([4]) that they term "Temporal Entropy" (TE). This indirectly measures how large the intervals generated are, smaller, and therefore better, ones yielding smaller TE. Different Wikipedia articles have different spans and time granularities, and therefore TE varies greatly among them. For example, a war is usually measured in years while a battle is measured in days. An event whose interval spans a few months in a war article should not be penalized the way that span should be in a battle article. It is then be necessary to normalize the TE. To do this, we divide the length of the event's interval in seconds by the length in days of the unit that is found for display on the timeline as described in the visualization section.

Temporal entropy does not give all the information, however. Spot-checking of articles reveals that many events - particularly those whose times were estimated - are not in the correct interval at all. A useful but impractical additional test, which we have not performed, would be human examination of the results to tell whether each event's assigned interval includes or is included in its actual interval. Then those that fail this test could be given maximum temporal entropy, to integreate this test's results into the former results.

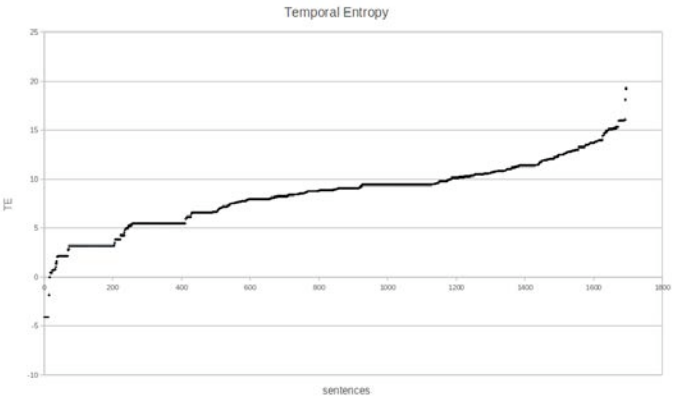


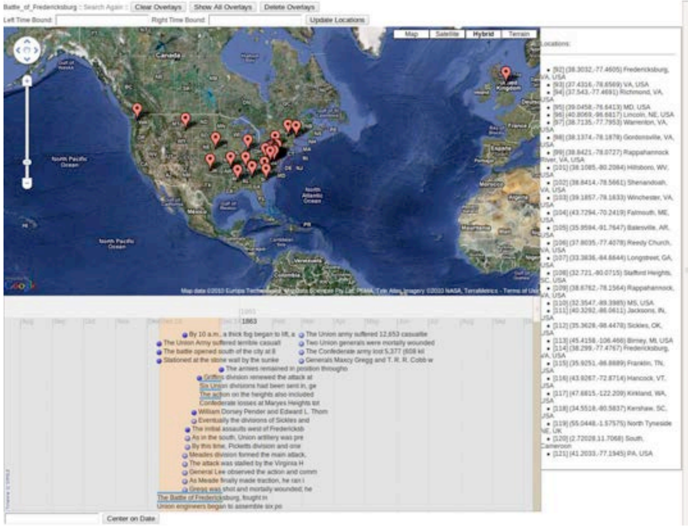
Figure 4: Temporal Entropy graph - the temporal entropies were sorted in increasing order for plotting. Temporal entropy is in log(seconds/days)

V. Web Visualization and Relation to Geospatial and Other Named Entities

A website is maintained where a user can view already processed Wikipedia articles or request the processing of new ones. The website allows users to view several aspects of the article. An example of what a user sees is in Figure 5. One aspect is the timeline of events whose creation has been the subject of this paper. Clicking any event on the timeline displays an infobox with further information about it, including the full sentence. The other major aspect is a map (via the Google Maps API) and list of locations the article mentions. These locations are processed using a separate system, and are the results of running a named entity recognizer (currently the Stanford NER) on the article and then geocoding and disambiguating the results. People and organizations, as extracted by the named entity recognizer, are also recorded. The locations, people, and organizations keep track of which sentences they are associated with, and can therefore be related to the timeline events. An example of this is seen in Figure 6, which shows a sample event infobox that includes associated locations, people, and organizations. If the user clicks on a listed location, the map centers on and zooms to that location. If the user clicks a location marker on the map, an infobox with the location's name and original context is displayed, as seen in Figure 7. Pages and their links to events and entities are stored in a database; the text is stored in one place and upon a user's request to view or process an article, the server generates the necessary data.

For the visual representation of the timeline, we use the Simile Timeline software, which allows the inclusion of a timeline on a website. The timeline is set up in javascript, which specifies the number of bands and their paramters, such as scale and center date. It also allows highlighting and magnification of certain regions ("hot zones"). These features are crucial to a readable timeline. Data is provided, in our case, in an XML format. Both the javascript and XML depend on the article, and they are generated upon a user's request to view a timeline. Most of the expensive computing is done when putting an article's information into the database, so a request to just view is fast.

Events that contain time expressions are simple to plot, as they have always a given start and end date. The events whose endpoints were guessed are placed in the interval they were given but are displayed at a subinterval that keeps them in order with respect to other such events in that interval. Then, the times that the timeline shows are far from certain, but the order is more likely to be correct.

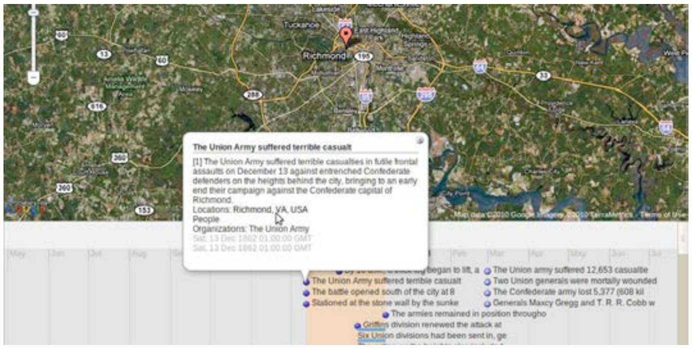


- By 10 a.m., a thick fog began to lift, a
- The Union Army suffered terrible casualt
- The battle opened south of the city at 8
- Stationed at the stone wall by the sunke
- The Union army suffered 12,653 casualtie
- Two Union generals were mortally wounded
- The Confederate army lost 5,377 (608 kil
- Generals Maxcy Gregg and T. R. R. Cobb w
- The armies remained in position througho
- Griffin's division renewed the attack at
- Six Union divisions had been sent in, ge
- The action on the heights also included
- Confederate losses at Maryes Heights tot
- William Dorsey Pender and Edward L. Thom
- Eventually the divisions of Sickles and
- The initial assaults west of Fredericksb
- As in the south, Union artillery was pre
- By this time, Picketts division and one
- Meades division formed the main attack.
- The attack was stalled by the Virginia H
- General Lee observed the action and comm
- As Meade finally made traction, he ran i
- Gregg was shot and mortally wounded; he
- The Battle of Fredericksburg, fought in
- Union engineers began to assemble six po

Locations:

- [92] (38.3032,-77.4605) Fredericksburg, VA, USA
- [93] (37.4316,-78.6569) VA, USA
- [94] (37.543,-77.4691) Richmond, VA, USA
- [95] (39.0458,-76.6413) MD, USA
- [96] (40.8069,-96.6817) Lincoln, NE, USA
- [97] (38.7135,-77.7953) Warrenton, VA, USA
- [98] (38.1374,-78.1878) Gordonsville, VA, USA
- [99] (38.8421,-78.0727) Rappahannock River, VA, USA
- [101] (38.1085,-80.2084) Hillsboro, WV, USA
- [102] (38.8414,-78.5661) Shenandoah, VA, USA
- [103] (39.1857,-78.1633) Winchester, VA, USA
- [104] (43.7294,-70.2419) Falmouth, ME, USA
- [105] (35.9594,-91.7647) Batesville, AR, USA
- [106] (37.8035,-77.4078) Reedy Church, VA, USA
- [107] (33.3836,-84.6844) Longstreet, GA, USA
- [108] (32.721,-80.0715) Stafford Heights, SC, USA
- [109] (38.6762,-78.1564) Rappahannock, VA, USA
- [110] (32.3547,-89.3985) MS, USA
- [111] (40.3292,-86.0611) Jacksons, IN, USA
- [112] (35.3628,-98.4478) Sickles, OK, USA
- [113] (45.4158,-106.466) Birney, MI, USA
- [114] (38.299,-77.4767) Fredericksburg, VA, USA
- [115] (35.9251,-86.8689) Franklin, TN, USA
- [116] (43.9267,-72.8714) Hancock, VT, USA
- [117] (47.6815,-122.209) Kirkland, WA, USA
- [118] (34.5518,-80.5837) Kershaw, SC, USA
- [119] (55.0448,-1.57575) North Tyneside NE, UK
- [120] (2.72028,11.7068) South, Cameroon
- [121] (41.2033,-77.1945) PA, USA

Figure 5: Example of an article's visualization



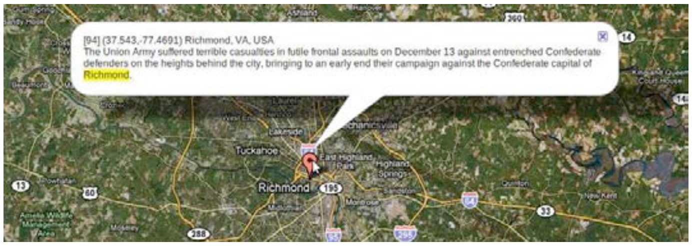
The Union Army suffered terrible casualt

[1] The Union Army suffered terrible casualties in futile frontal assaults on December 13 against entrenched Confederate defenders on the heights behind the city, bringing to an early end their campaign against the Confederate capital of Richmond.
Locations: Richmond, VA, USA
People
Organizations: The Union Army
Sat, 13 Dec 1862 01:00:00 GMT
Sat, 13 Dec 1862 01:00:00 GMT

- By 10 a.m., a thick fog began to lift, a
- The Union Army suffered terrible casualt
- The battle opened south of the city at 8
- Stationed at the stone wall by the sunke
- The Union army suffered 12,653 casualtie
- Two Union generals were mortally wounded
- The Confederate army lost 5,377 (608 kil
- Generals Maxcy Gregg and T. R. R. Cobb w
- The armies remained in position througho
- Griffin's division renewed the attack at
- Six Union divisions had been sent in, ge

Figure 6: Event infobox in the visualization, after clicking "Richmond, VA, USA"

The identification of "hot zones" is an important part of the process of visualization, because the articles often give some background and after-effects of their main events, but most of the sentences cluster in one or more important intervals. We originally only created one hot zone and made it the "middle of the article times" interval described previously. This was not optimal, because times often cluster into multiple, densely populated sections. To generate the endpoints for more than one hot zone, we sort the times for an article and examine the time differences (in days) between consecutive times. Long stretches of low differences indicate dense time intervals. An example of these differences graphed for an article with obvious clusters is shown in Figure 8. Figure 9 shows part of its corresponding timeline. It is not necessary for the differences to be zero, just low, so a measure was needed for how low was acceptable. We chose to remove outliers from the list of time differences and then take the average of the new list. To remove outliers, we proceed in a common manner and calculate the first and third quartiles ($Q1$ and $Q3$) and then remove values greater than $Q3 + 3 * (Q3 - Q1)$ or less than $Q1 - 3 * (Q3 - Q1)$. The latter quantity was usually zero, so this ended up removing high outliers. This average was the threshold below which a difference was called low. We also had to ensure that too many hot zones were not generated, so we chose a threshold for the number of consecutive differences that had to be low for the interval to be a hot zones; we chose 10, based on what we felt would be appropriate for a timeline.



[94] (37.543,-77.4691) Richmond, VA, USA
The Union Army suffered terrible casualties in futile frontal assaults on December 13 against entrenched Confederate defenders on the heights behind the city, bringing to an early end their campaign against the Confederate capital of Richmond.

Figure 7: Geospatial entity infobox in the visualization

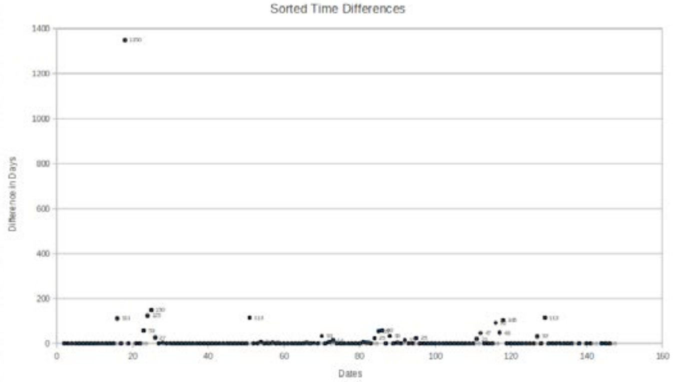
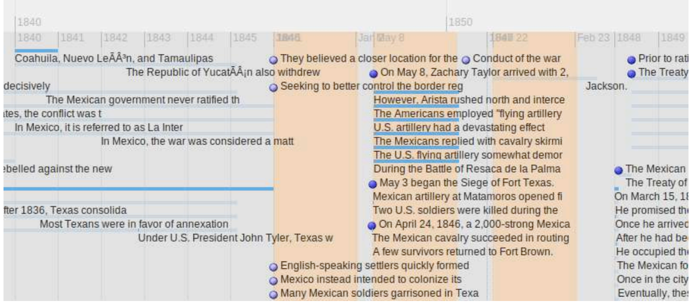


Figure 8: Differences between consecutive times in the sorted time list for Mexican-American War



Coahuila, Nuevo LeÃÂ³n, and Tamaulipas
The Republic of YucatÃÂ¡n also withdrew
decisively
The Mexican government never ratified th
ites, the conflict was t
In Mexico, it is referred to as La Inter
In Mexico, the war was considered a matt
ebelled against the new
fter 1836, Texas consolida
Most Texans were in favor of annexation
Under U.S. President John Tyler, Texas w
They believed a closer location for the
Seeking to better control the border reg
English-speaking settlers quickly formed
Mexico instead intended to colonize its
Many Mexican soldiers garrisoned in Texa
Conduct of the war
On May 8, Zachary Taylor arrived with 2,
However, Arista rushed north and interce
The Americans employed "flying artillery
U.S. artillery had a devastating effect
The Mexicans replied with cavalry skirmi
The U.S. flying artillery somewhat demor
During the Battle of Resaca de la Palma
May 3 began the Siege of Fort Texas.
Mexican artillery at Matamoros opened fi
Two U.S. soldiers were killed during the
On April 24, 1846, a 2,000-strong Mexica
The Mexican cavalry succeeded in routing
A few survivors returned to Fort Brown.
Jackson.
Prior to rati
The Treaty
The Mexican
The Treaty of
On March 15, 18
He promised th
Once he arrived
After he had be
He occupied th
The Mexican fo
Once in the city
Eventually, the:

Figure 9: Timeline for Mexican-American War, containing 3 hot zones

VI. Conclusions

The difficulty in classifying important events may lie with the features we chose or with the idea that one can even classify events as "important" or "not important." From the performance improvement for both machine and humans when switching from words to sentences, it seems that it is much more natural to ask about important sentences. This moves into the territory of text summarization, and it might be more pertinent to rank the sentences than to put them into two classes.

The temporal relation extraction turns out to work fairly well for this domain of historical narratives because the sentences are often ordered in the text similarly to their temporal order. In another domain, even one rich in temporal information like biographies, it might not do as well. Further, due to the algorithms for anchoring times to years and for giving times to sentences without them, errors tend to build up and carry over a lot; some kind of check or reset condition, if one could be developed, would have helped accuracy.

Finding and displaying links among events, people, organizations, and locations succeeded since the events are just considered sentences. The visualization of this has good usability and is a useful tool to people analyzing the articles.

VII. Future Work

The mediocre performance of the important event classifier could be improved upon in the future. It is possible that more features would help it better classify. Ideas for sentence features that were not implemented included calculating the current features for the adjacent sentences. Classifiers other than SVMs were also considered, and this option could be explored. Since the task is similar to text summarization, methods like TF/IDF that are used for summarization could be adapted to one document and tried out. It is also possible that the definition of "important" is not objective or specific enough, and further investigation into this would be useful.

While classifying more events positively tends to increase scores, visualizing so many is not desirable. To reduce the number that get selected, we could reduce the number that are classified at all by doing preprocessing on the sentences. We suggest running TextRank on the sentences originally and then picking some fraction of the top ranked sentences to classify.

The classifier also takes more time than is desirable to calculate the features. Long articles can take 2-3 minutes, and since these articles are being processed while users wait for them, the current run times are not optimal. Using multithreading on processing articles would make some improvement. There are three specific features that take the most time to calculate - similarity, named entity weight, and TextRank rank. Experimenting with removing any of these features while preserving accuracy could make a more efficient classifier.

The place for the most improvement is likely the temporal portion of the process. Most of the algorithms we use here are naive and do not make use of existing tools. The regular expression extractor works fairly well, but needs expansion to handle relative expressions like "Two weeks later, the army invaded." To implement this with the same accuracy that the extractor currently has should not be hard, as when the expression is recognized, the function that it implies (in this example, adding two weeks) can be applied to the anchoring time. It also cannot currently handle times B.C.E. More difficult would be a change to the program to have it extract multiple expressions per sentence (beyond the range-like expressions it already finds). If events are ever considered smaller than sentences, this is crucial. Having events smaller than sentences would also allow the use of Ling and Weld's Temporal Information Extraction system ([4]), which temporally relates events within one sentence.

The visualization could also be improved with some changes to Simile's code for displaying a timeline, particularly the width of the timeline, which does not automatically adjust for more events. Because the important event classifier positively identifies a lot of sentences that are often close in time, the listing of events on the timeline overflows the space, so more must be allotted even for sparser articles. This could be improved with automatic resizing or zooming. Another improvement would be to create event titles that better identify the events since they are currently just the beginnings of the sentences. Finally, more features could be added to the visualization, especially in terms of filtering by time or location range.

References

[1] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, November 1983. [Online]. Available: http://dx.doi.org/10.1145/182.358434

[2] L. Zhou, G. B. B. Melton, S. Parsons, and G. Hripcsak, "A temporal constraint structure for extracting temporal information from clinical narrative." *J Biomed Inform*, September 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jbi.2005.07.002

[3] M. Verhagen and J. Pustejovsky, "Temporal processing with the tarsqi toolkit," in *22nd International Conference on on Computational Linguistics: Demonstration Papers*. Manchester, United Kingdom: Association for Computational Linguistics, August 2008, pp. 189–192.

[4] X. Ling and D. Weld, "Temporal information extraction," in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-10)*, Atlanta, GA, July 2010.

[5] R. Swan and J. Allan, "Extracting significant time varying features from text," in *CIKM '99: Proceedings of the eighth international conference on Information and knowledge management*. New York, NY, USA: ACM, 1999, pp. 38–45. [Online]. Available: http://dx.doi.org/10.1145/319950.319956

[6] D. Ahn, "The stages of event extraction," in *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*. Sydney, Australia: Association for Computational Linguistics, July 2006, pp. 1–8. [Online]. Available: http://www.aclweb.org/anthology-new/W06/W06-0901.bib

[7] R. Saur, R. Knippen, M. Verhagen, and J. Pustejovsky, "Evita: a robust event recognizer for qa systems," in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, October 2005, pp. 700–707.

[8] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet::similarity: measuring the relatedness of concepts," in *HLT-NAACL '04: Demonstration Papers at HLT-NAACL 2004 on XX*. Morristown, NJ, USA: Association for Computational Linguistics, 2004, pp. 38–41.

[9] R. Mihalcea and P. Tarau, "Textrank: Bringing order into texts," in *Proceedings of EMNLP 2004*, D. Lin and D. Wu, Eds. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411.

[10] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.5547

[12] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *Machine Learning: ECML 2004*, 2004, pp. 39–50. [Online]. Available: http://www.springerlink.com/content/pa57eam5t5dkem4h

Improving Ontology Alignment Accuracy with Detailed Feature Extraction Tools

Josh D. Dispoto

Abstract—**Ontologies in the biomedical field have massive collections of information that are difficult to align and compare. Outlined in this paper are strategies to analyze current state of the art ontology alignment algorithms by analyzing results and performance with both small- and large-scale ontologies. Results of testing various sets of ontologies against a few named Ontology alignment tools is also included. The ultimate goal will be the modification and optimization of the Stoutenburg algorithm with Branch and Bound in order to produce superior results in precision and run time for large scale ontology alignment.**

I. Introduction

IN recent years, several algorithms for ontology alignment have been developed. The algorithms are needed for the integration and optimization of the many ontologies now available on the web [1]. Similarly, the size and scale of the ontologies available, as well as the potential alignments between said ontologies has grown substantially [2].

Ontologies are always being updated and revised. Ontology alignments also occur much more. These actions of ontological expansion inevitably lead to larger and larger ontologies[1]. Also, creating large ontologies takes time, funding, and can often be very difficult to create [3]. The clearly visible advantages to aligning ontologies manually include highly accurate alignment results. Therefore, to reduce the time of aligning ontologies, a highly accurate alignment program can be used.

The intention of this paper is to describe the approach used to extract and use highly detailed features from ontologies as a basis for alignment. Also, a detailed description of the algorithm currently in development that utilizes the aforementioned feature extraction tools.

II. Problem Statement

The goals of this project include: determining new, detailed features to extract from ontology concepts to form a basis for ontology alignment as well as developing an ontology alignment algorithm that will not only accurately extract the features but also use them to determine highly accurate alignments amongst ontology concepts.

The vast amount of information included within the ever-growing ontologies today has become increasingly difficult to analyze and compare. The algorithms being used today to align ontologies are all unique in their own strategies to determine matching ontology concepts. However, some more than others rely on an approach that may address reducing the run time of ontology alignment slightly more than accuracy[4]. Similarly some programs have sacrificed accuracy to improve specific concepts of their algorithm. The end result is the primary focus, however. Without accurate results, the whole purpose of ontology alignment falls from view. In order to uphold the highest standard of quality in information present in ontologies, the highest level of accuracy must be achieved in the process of aligning ontologies.

III. Related Research

Among the several ontology alignment algorithms and tools, different methods for alignments have been developed. These automated systems utilize a broad range of alignment approaches such as logical axiom similarity processing mentioned in [5] and similarity grouping followed by detailed relationship comparisons as in [2].

However, the Stoutenburg algorithm steps away from similarity-based alignments and uses the knowledge of upper ontologies and applies support vector machine (SVM) technology to produce non-equivalence relation alignments[4].

The Stoutenburg approach uses WordNet[1] and OpenCyc[2] to analyze concepts of the ontologies being aligned [4]. These tools extract features within the ontologies and organizes them appropriately. The resulting classifications are passed through a Support Vector Machine which determines proper alignments [4]. However, it fell short in execution time pushing upwards of 96 hours to align ontologies [4].

In response, paths were taken to decrease execution time of the Stoutenburg algorithm which led to the Branch and Bound algorithm. As described in [4] the Branch and Bound algorithm uses a concept of one ontology and determines semantic closeness to concepts in a second ontology; if it was determined to not have semantic closeness, the concept of ontology will be taken out as well as its children, otherwise the concept pair would continue to be aligned. Also, synonymous, hyponymous, and hypernymous relations are used to determine semantic closeness. Based on the results in [4] hyponymy yielded the best results. This later approach, however, yielded less accurate results and as a result still has room for improvement[4].

Altogether, there have been many approaches to ontology alignment. However, only recently have more approaches incorporated ways to handle large ontologies. Among the small number of programs accounting for scalability, the Stoutenburg branch and bound algorithm shows the most promising preliminary results in reducing run time. This

□Paper submitted July 30, 2010. Submitted as an initial proposal draft as part of the REU NSF Internship program.

[1] http://wordnet.princeton.edu/

[2] http://www.cyc.com/opencyc

further exemplifies the need for higher accuracy in ontology alignments as the topic of scalability is an area of increasing focus. The accuracy of alignments must be maintained and improved for greater end results.

IV. FEATURE EXTRACTION

When addressing the aspects that might increase accuracy, we focused on the features being extracted from each ontology concept. Based on the prior work mentioned in [4] we found that the features being extracted included the following:

- Linguistic-based
- Pattern-based
- String-based
- Extrinsic-based
- Bag-of-words-based

Linguistic-based features as mentioned in [4] include those meaningful prefixes in strings that might indicate a subclass relationship because the algorithm in [4] uses an SVM to find class correspondence. Pattern-based features are similar to linguistic features in that they are simply patterns that arise in ontology concept names.

String-based features are those that include sub-string comparisons[4]. Finding that some ontology class pairs had the same ending words made for some meaningful data.

Bag-of-words-based features consist of the scoring of synonyms, hyponyms, and hypernyms among ontology concept pairs[4].

Finally, Extrinsic-based features are those concept pairs that hold syntactic or semantic relationships based on outside knowledge sources such as WordNet and OpenCyc as mentioned earlier[4].

Altogether these features are essential for accurate ontology alignments. All the preceding features were used in their own specific way in [4] to retrieve results using the Stoutenburg algorithm.

However, we used this knowledge to create our own specific set of features that we use in our own specific implementation of an ontology alignment algorithm.

V. CONCEPT

Based on the domain we will be focusing our efforts – biomedical ontologies – we found that there is a lot of information within the class names and labels of the ontologies. Our idea is to extract this information and use it as a basis for the concept alignment within the ontologies.

More specifically, the linguistic value of each class name or label is our focus. For example, the following label found in the Gene Ontology holds several key words that accurately describe what that specific class represents:

“The chemical reactions and pathways involving amino acids containing sulfur, comprising cysteine, homocysteine, methionine and selenocysteine.”

Obviously there is a lot of information within the label that gives a vivid representation of the ontology class. However, while the linguistic-based features mentioned in [4] focused only on prefixes for certain relationships, our method uses not only prefixes, but also the root and suffixes of those keywords.

With all the affixes and roots compiled as a large knowledge base for comparisons, we can make more accurate results because of the highly specific information we extract.

However, there is still the hurdle of designing an algorithm that will accurately extract the linguistic features we seek.

VI. AFFIX/ROOT EXTRACTION

The goal of this feature extraction API is to take in a string word and extract all the prefixes, suffixes, and root from the word which can then be used by the user for whatever reason they may have for it.

We utilize character manipulation to complete this process. We will be using the word “invisible” as an example to provide a visual reference for how this algorithm works.

The extraction algorithm goes in the following order in extracting information from the word:

(1) extract suffix
    (1.1) extract additional suffixes
(2) extract root
(3) extract prefix
    (3.1) extract additional prefixes

Therefore, to begin with the algorithm strips off one character at a time from the front of the string and compares the newly truncated string against a large collection of suffixes that have been compiled over time into one file.

Figure 1 details the process through a more visual representation.

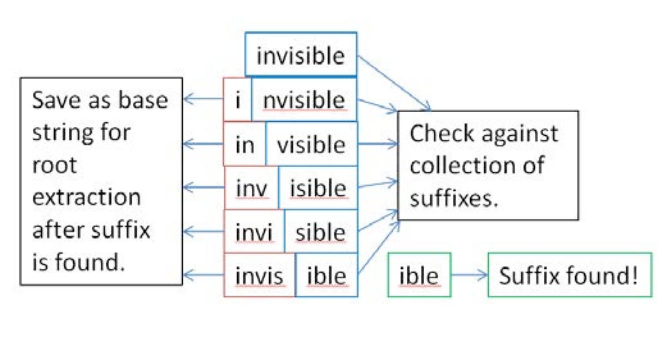

invisible
Save as base string for root extraction after suffix is found.
i nvisible
in visible
inv isible
invi sible
invis ible
Check against collection of suffixes.
ible
Suffix found!

Figure 1: Suffix extraction process

The suffix extraction process iterates until no more suffixes are found. This ensures that those words that have multiple suffixes are taken care of. However this iterative process can also cause inaccurate extractions but that is detailed later.

Following suffix extraction, the saved base string – in this case the “invis” string – is passed to the root method to extract the root of the word, if there is one.

The root extraction process mirrors that of the suffix extraction except that there are no iterations, there is only one pass through the string to determine a root. Figure 2 shows the continued extraction process of the word "invisible".

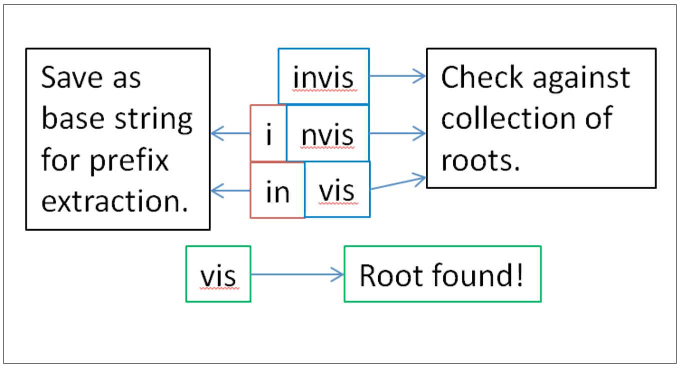

Save as base string for prefix extraction.
invis
i
nvis
in
vis
Check against collection of roots.
vis
Root found!

Figure 2: Root extraction process

Again, the root goes by the same process as the suffix extraction process, just without iterations. In some cases, however, there may be no root found. This may be caused because the root within the word is not in the collection of roots or it may be because there is no root in the word and the word consists of only prefixes and suffixes. Either way, the next step is to extract the prefix of the word.

The prefix extraction process differs slightly. Instead of stripping characters from the front of the string, the prefix extraction algorithm takes characters from the end of the string. This way, in case no root was found and there were excess characters on the end of the string, it will not interfere with the prefix process comparing the string to the collection of prefixes. Figure 3 shows the final results of the extraction process of "invisible". However, to further detail how the prefix extraction algorithm works, see Figure 4.

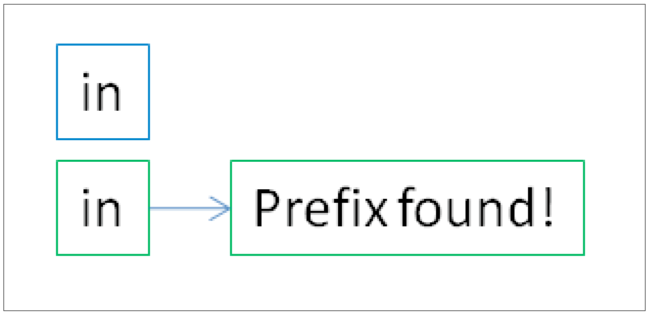


Figure 3: Prefix extraction process

The string "in" that remained was quickly checked and found within the collection of prefixes and so the extraction process for "invisible is complete.

For the following example in Figure 4, we will be using the word "covalently". Assume that the suffix "ly" has already been stripped from the string and that no root was found when checked through the root extraction process.

In this case, the left behind string will be iterated again until no prefix is found. Also, just like the suffix extraction process the iterations can cause inaccuracies to occur. However there have been steps taken to error check the validity of the end results which will be mentioned later.

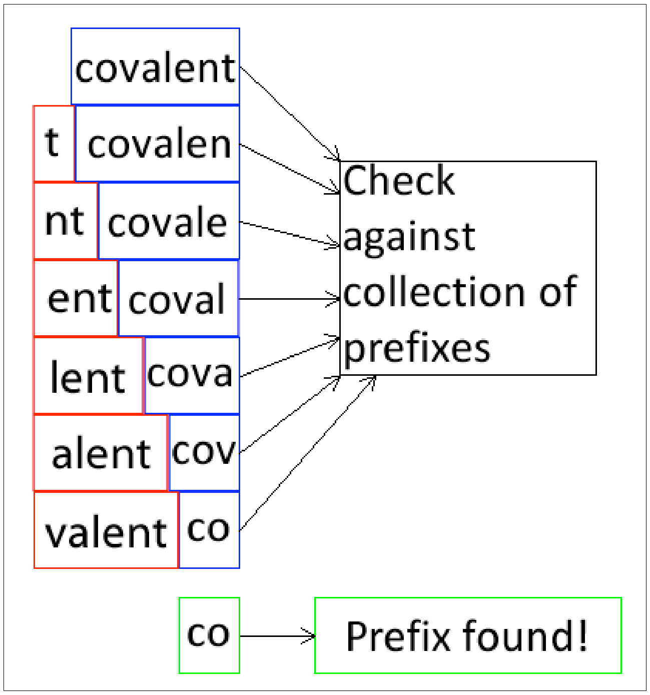


Figure 4: Complete Prefix extraction process

After all suffixes, prefixes and the root have been extracted the process is nearly complete. For the most part these three methods will accurately extract all the information from the string and will be available to the user to do with as they wish.

All that remains are some error checking procedures that handle some of the inaccuracies that may occur.

VII. ERROR CHECKING

Language, especially the English language, has many exceptions that have the potential to fool the extraction algorithm detailed above. Therefore we have implemented a few error checking algorithms to ensure the accuracy of the affix and root extraction process.

To begin, before the string is passed through any extraction algorithms it is checked for proper spelling and whether or not it is an accurate word in a large dictionary file. Currently the file has over 180,000 word entries that provide an adequate baseline. There is still room to expand the dictionary with more biomedical terms that will cover much of the domain we will be testing in.

Once the word has passed the spell-checking test, it moves through the extraction methods and based on the ending results of the extraction process will be put through further error checking algorithms.

There are special cases that can cause double checking to occur. Some of the cases that can cause inaccuracies in the extraction process include:

- Too many suffix extraction iterations that strip away substrings that are not accurate suffixes

    - Ex. “covalently” on the first run through will find “ly”, “ent”, and “al” as suffixes.
    - “ly” is the only suffix in this word
- Inaccurate root extraction
    - Ex. In the case of “” the root was extracted as “” when it should accurately be “”.
- No root found
    - Leaves behind remaining string as in Figure 4
- Too many prefix extraction iterations that identify incorrect prefixes if no root was found.
    - Ex. If no root was found and the prefix extractor stripped out a registered prefix still included in the extra substring

The first error check algorithm is used if there is a string left over from the extraction process as in Figure 4. The algorithm takes the left over substring and begins reconstructing the word by adding the last suffix found, one character at a time.

An example of this process can be found in Figure 5. Assume in Figure 5 that the word “covalently” is used. Also, assume that the suffix iteration extracted incorrect suffixes as mentioned above in the first bullet of inaccuracy cases and that the remaining string consists of only “v” because the prefix was extracted as well.

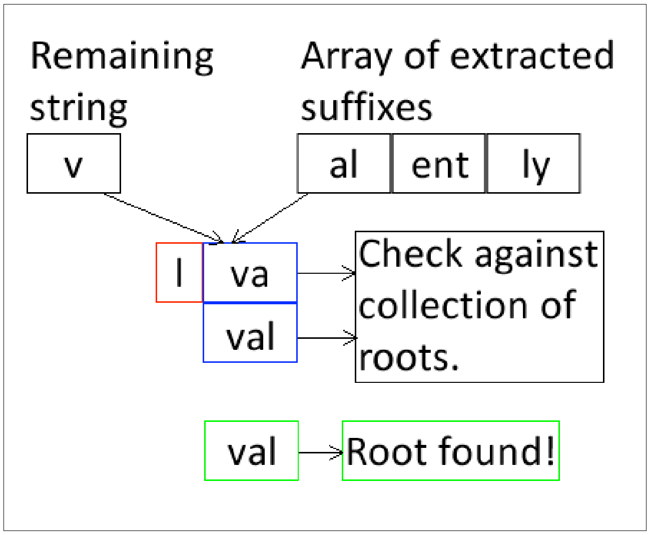

Remaining string
Array of extracted suffixes
v
al
ent
ly
l
va
val
Check against collection of roots.
val
Root found!

Figure 5: Error check - phase 1 process

If the error check finds no roots after reconstructing the first time, it will continue checking by taking the reconstructed string (in this case “val”) and adding the latest prefix that was extracted. Therefore, the string would be further reconstructed to “coval”. This substring is then checked against the prefix collection to make sure that the correct prefix was extracted and that the over-iteration of suffix extractions did not cause an incorrect prefix to be found.

Also, during this initial error-check, if any inaccuracies are found, the prefix, root and suffix information will be altered immediately to reflect the correct extractions.

Once the first error check is complete the next error case is reviewed. The occurrence may arise in which there is no string remaining after all extraction processes have completed but there are no prefixes found. If this is the case and there are more than three suffixes that have been extracted, a new error check is performed. The first step it takes is to send the root word through the prefix extraction to ensure that there was not a premature extraction that caused no prefixes to be found. After finding finishing the prefix extraction method, this method reconstructs the string starting with the root and adding each suffix in order. Each time a suffix is added to the remainder string, it will be passed through the root extraction method to ensure accuracy in the root word.

After all checks have concluded, the proper changes are made and all error checking is complete.

VIII.IMPLEMENTATION

While the extraction algorithm may seem to accurately extract linguistic features from words, we could only do so much to test its level of effectiveness. We created several test cases to ensure the accuracy of the extraction algorithm. However, we decided to test it further by developing a unique ontology alignment program that utilized this tool.

The program is separated into four sections based on the information we would use to determine equivalence between ontology classes:

- Ontology class labels
- Affix/root definition closeness database
- The ontologies
- Alignment process

The information we will be focusing on to utilize the extraction algorithm are the labels for each ontology class. As described in Section V. CONCEPT, the labels of an ontology class hold a large amount of information that, if extracted properly, can serve to be an excellent source for highly accurate ontology alignments.

The ontology class labels section of the program holds the entire affix and root information as well as the definition closeness data from the database which will be covered soon.

The class labels code takes in the label string and separates the string into keywords. The keywords are then passed through the extraction algorithm where all the affix and root information is saved for later use. Once all the affix and root information has been collected, they are all passed through a database specific identification numbers are collected in return for use in the alignment process.

The database is a large knowledge base similar to that of WordNet and other defining database tools. The database holds synsets of data that carry an identification number. Each synset also holds a definition in which specific affixes or roots can be linked to. For example, the synset with the definition “one” also has specific affixes linked to it such as “uni” and “mono” which hold the same definition. This specific synset may hold the identification number of “10”. Therefore, if a synset of similar definition is present in the database such as "single" the identification number would be close to the "one" synset and possibly be "11" (See Figure 6). The closeness of definitions is the basis for our alignment process.

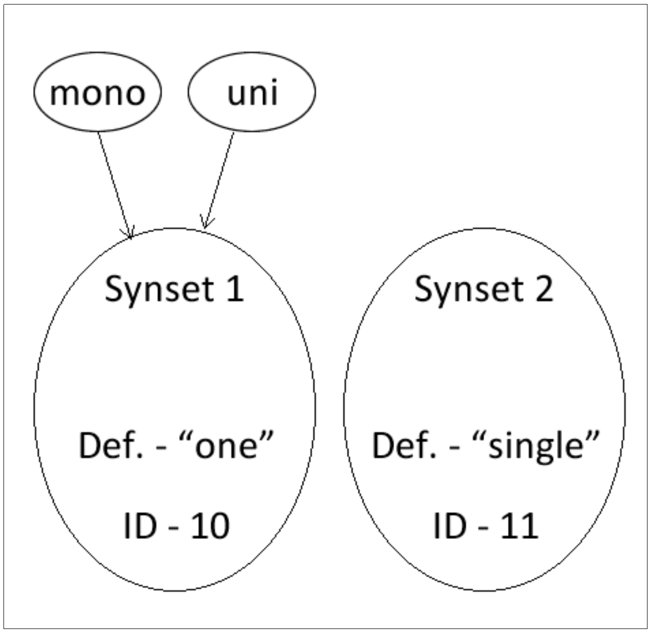


Figure 6: Database layout example

The next section includes the ontologies themselves. Two ontologies are passed two the alignment process to begin. Therefore, each ontology object holds all the class objects within that ontology as well as the class label objects. The ontology object also can return the specific URI of a class for alignment output.

Finally, the alignment process uses all the aforementioned information to determine equivalence and similarities between concepts from each ontology.

As described earlier, each class holds a label and within each label object is a collection of synset ID numbers from the definition database. The alignment process compares each label with the labels in the second ontology by calculating an accuracy value. Figure 7 shows how the process works for each comparison.

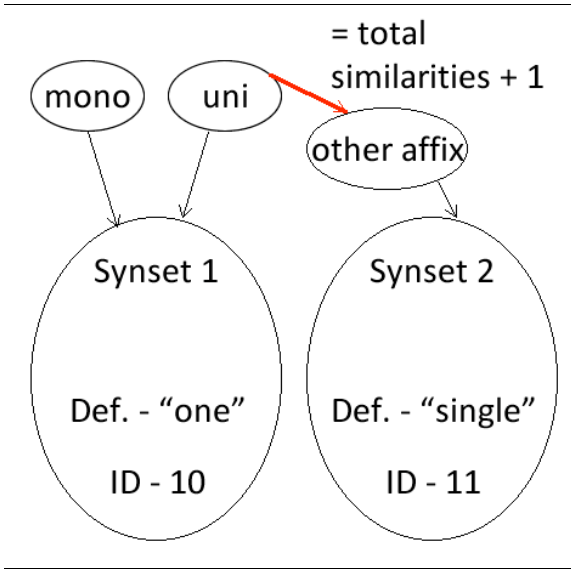


Figure 7: Synset Comparison and scoring

For every synset ID in Label 1 that is similar to a synset ID in Label 2, a counter is incremented and by the end of all the comparison the total calculated similarities is divided by the total number of comparisons made. A decimal such as this is calculated for all the prefix, suffix and root synset ID comparisons. Next, all the decimal values are added together and averaged for a final accuracy value.

The final accuracy is checked against a predetermined accuracy threshold. If the result is higher than the threshold, the both class URIs are printed to a text file with the accuracy value all separated by a semicolon. The results below the threshold point can also be printed if the user decides to use the information.

Also, a timer has been implemented to record the run time of the alignment process. The run time is displayed in the results both above and below threshold in milliseconds.

In conclusion, the alignment program utilizes the linguistic features derived from the word extraction algorithm in hopes of highly accurate results and excellent alignment results.

References

[1] S. K. Stoutenburg. *Acquiring advanced properties in ontology mapping.* (2008). PIKM '08: Proceeding of the 2nd PhD workshop on Information and knowledge management. pp. 9-16.

[2] Wei Hu, Yuzhong Qu, Gong Cheng. (2008). Matching large ontologies: A divide-and-conquer approach. *Data & Knowledge Engineering*. Vol. 67 (1). pp. 140-160

[3] R. Freckleton. (2010). *Ontology Alignment Using Automatic Machine Learning Techniques.*

[4] S. K. Stoutenburg, "Advanced ontology alignment: New methods for biomedical ontology alignment using non-equivalence relations," PhD. Dissertation, University of Colorado at Colorado Springs, 2009.

[5] M. Ehrig, Y. Sure. (2005). *Ontology Mapping by Axioms (OMA).* Professional Knowledge Management: Lecture Notes in Computer Science. Vol. 3782/2005. pp. 560-569

Designing Soft Keyboards for Brahmic Scripts

Lauren Hinkle

Abstract

Soft keyboards, because of their ease of installation and lack of reliance on specific hardware, are a promising solution as an input device for many languages. Developing an acceptable soft keyboard requires frequency analysis of characters in order to design a layout that minimizes text-input time. This paper proposes using various development techniques, layout variations, and evaluation methods for soft keyboard creation for Brahmic scripts. We propose that using optimization techniques such as genetic algorithms to develop multi-layer and/or gesture keyboards will increase the speed at which text can be entered.

1 Introduction

In an increasingly fast paced world, being able to input text quickly is important to all users. Standard Roman-alphabet based languages have been studied in detail, and it is important to now develop efficient keyboards in languages that have not been highly researched. Many Indic languages have only rudimentary keyboards that were developed so that users could input text, not with the goal of optimization. As a result, many of these keyboards have not had a significant impact because they are difficult to learn and inefficient use.

Soft keyboards allow a user to input text with and without a physical keyboard. They are versatile because they allow data to be input through mouse clicks on an on-screen keyboard, through a touch screen on a computer, cell phone, or PDA, or by mapping a virtual keyboard to a standard physical keyboard. With the recent surge in popularity of touchscreen media such as cell phones and computers, well-designed soft keyboards are becoming more important everywhere.

For languages that don't conform well to standard QWERTY-based keyboards that accompany most computers, soft keyboards are especially important. Brahmic scripts (Columas et al. 1990) have more characters and ligatures than fit usably on a standard keyboard. A soft keyboard allows any language to have custom layouts based on the frequency of character and ligature use within that language that do not conform to a standard keyboard. While physical keyboards have been designed for Brahmic scripts, such as (Joshi et al. 2004), they are cumbersome and difficult to learn. The development and spread of soft keyboards would allow the use of an easier to learn and more efficient keyboard.

Web service providers such as Google.com and Wikipedia.com have developed soft keyboards for Brahmic languages, but these are not optimized. Computer users are often forced to use English keyboards to tediously type their script. This limits people's ability to use computers and thus connect themselves to the many resources computers can provide. The development of soft keyboards for these languages has the dual benefit of allowing more people to be able to use a computer in their native language and to prepare users for touch-screen technologies.

Developing a soft keyboard for a Brahmic script is important, but in order for it to be usable it is necessary to design it such that it is easily learned as well as efficient.

An efficient soft keyboard must optimize the speed with which a user can input text using only one input device (for example a stylus, finger, or mouse). A user must be able to select the desired characters as quickly as possible. Brahmic scripts pose an important problem that must be overcome: the number of characters that should appear on the keyboard to maximize efficiency. Brahmic scripts have more characters than the standard Roman-based languages most people are familiar with. For example, the Eastern Nagari script has 37 consonants, 11 vowels, and 4 special characters that can be used individually as well as combined to create about 250 different ligatures.

In addition to deciding whether ligatures should appear on the keyboard, the layout of the characters must be designed with efficient text-entry in mind. More frequently used characters should be placed centrally, and characters that are often used together should be located close to one another. This would minimize the travel distance between characters being selected and therefore increase the input speed.

2 Related Work

While efficient soft keyboards have not been widely developed for Brahmic scripts, there has been much research in techniques for development of soft keyboards in English.

Soft keyboard layouts have developed from simple and familiar to seemingly arbitrary, yet designed by computers to be as efficient as possible. Early techniques for English soft keyboards used either alphabetic ordering or the traditional QWERTY layout. While these are still the most commonly used keyboards, great enhancements have been made in optimization. MacKenzie et al. were among the first to design a layout based on character frequency. They used Fitt's Law and trial-and-error hand-placement of frequently occuring bigraphs to develop the OPTI keyboard (MacKenzie and Zhang 1999). Keyboards were further improved upon by using computers to iterate over many solutions to computation-heavy algorithms from physics such as Hook's Law and a Metropolis random walk algorithm. Machines developed and tested increasingly efficient soft keyboards that were able to reach theoretic upper-bounds of text input of approximately 42 wpm (Zhai et al. 2000). The next major step in efficiency was undertaken by employing genetic algorithms. The keyboards generated using this technique were more efficient for every layout tested than any previous soft keyboard (Raynal and Vigouroux 2005). The impressive performance of genetic algorithms leads to the idea of using other optimization techniques for keyboard layout. While such techniques have not been applied to keyboards aimed at the general public, ant colony optimization strategies have been applied to virtual keyboards designed to make text input simpler for people with disabilities (Colas et al. 2008).

If these steps worked to create efficient soft keyboards for English, we hypothesize they will work as well for other languages.

Brahmic soft keyboards are currently at a point in their development analogous to where English soft keyboards were a decade ago. They are currently organized in traditional alphabetic ordering or in consonant-vowel groups. However, there have been several interesting techniques for input schemes employed. Although there are some designs for single layer keyboards, reminiscent of English soft keyboards (Sowmya and Varma 2009), this is not the trend in Brahmic keyboards.

As a result of the great number of characters in the alphabets, including diacritics, many Brahmic keyboard designers have opted for layered keyboards. A layered keyboard has multiple characters residing in the same location that are accessible by either hitting a button that toggles between layers (Rathod and Josh 2002) or rolling over base characters to reveal others (Shanbhag et al. 2002). Shanbhag's keyboard has a combination of these effects, with multiple layers that can be toggled between as well as a layer that has consonants grouped together and accessible by rolling over the group icon (Shanbhag et al. 2002).

Another technique to decrease the necessary number of keys is the use of gestures on the keyboard. In a gesture keyboard, the user draws any necessary diacritics for each character as they select it (Krishna et al.

2005). Gesture based keyboards have taken some of their ideas from English gesture-based keyboards such as T-Cube, in which a user selects different characters by flicking their mouse or stylus in different directions (Venolia and Neiberg 1994).

These techniques have the potential to increase the input speed of the user by decreasing the search time for characters and decreasing the necessary distance to travel between characters. However, no theoretic input speeds have been calculated for them. In addition, both techniques stand to benefit by reordering the keyboard by frequency of use of unigraphs and bigraphs, rather than alphabetically.

3 Brute-Force Brahmic Soft Keyboard Design

As a first attempt in designing a soft keyboard, we took a brute-force approach. We implemented a three-level pop-up menu based soft keyboard for Assamese, the Eastern-most Indo-European language and one that uses the Eastern Nagari variant of Brahmic scripts. The first level provided an individual key for each consonant (See Figure 1). It also had a single key for each of the following sets of characters: vowels, vowel diacritics, digits and special punctuation marks.

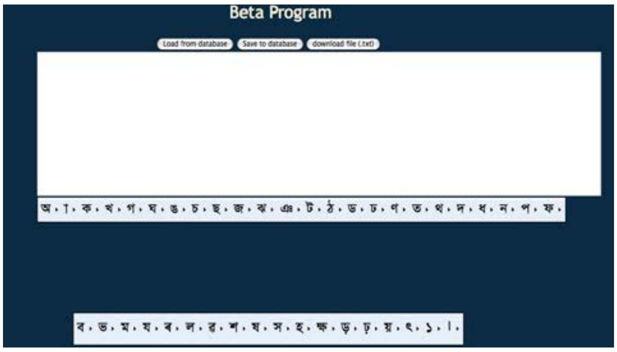

Beta Program
Load from database
Save to database
download file (.txt)
অ া ক খ গ ঘ ঙ চ ছ জ ঝ ঞ ট ঠ ড ঢ ণ ত থ দ ধ ন প ফ
ব ভ ম য ৰ ল ৱ শ ষ স হ ক্ষ ড় ঢ় য় ৎ ১ ।

Figure 1: The top-level menu items for character entry. The 11 vowels are represented by the first menu item in the top row. The second item in the top row represents the medial diacritic representations of the eleven vowels. The last three items on the bottom row represent special characters, numerals and punctuation, respectively.

The second level of keys were organized as follows. When the first vowel, , which appears on the initial screen, is pressed on, a horizontal pop-up menu with the 10 other vowels appeared. Similarly, when one presses on the first diacritic , the diacritics for the ten other vowels appear in a horizontal pop-up menu. When one presses on any of the consonants at the first level, every combination of the consonant with vowel diacritics and every ligature in which the consonant participates shows up. The second level menus are rectangular. An example of the menu that pops up for is shown in Figure 2.

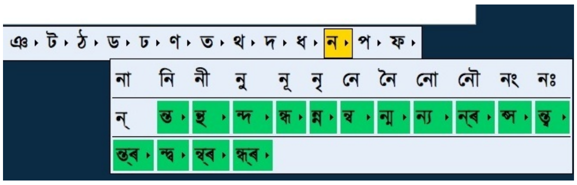

ঞ ট ঠ ড ঢ ণ ত থ দ ধ ন প ফ
না নি নী নু নূ নৃ নে নৈ নো নৌ নং নঃ
ন্
ন্ত ন্থ ন্দ ন্ধ ন্ন ন্ব ন্ম ন্য ন্ৰ ন্স ন্ত্ব
ন্ত্ৰ ন্দ্ব ন্থ্ৰ ন্ধ্ৰ

Figure 2: Second level menu (light blue in green) for a consonant.

There exists a third level of menus as well. Each ligature in the second layer has a sub-menu of all the vowel diacritics that can be added to it. The level 3 menu for a ligature is shown in Figure 3.

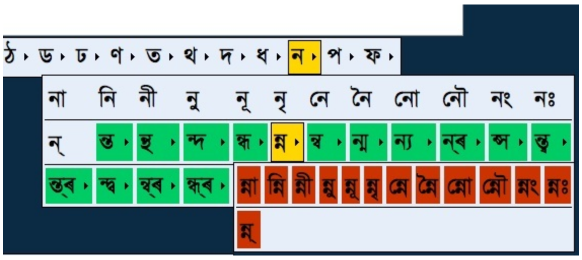

ঠ ড ঢ ণ ত থ দ ধ ন প ফ
না নি নী নু নূ নৃ নে নৈ নো নৌ নং নঃ
ন্
ন্ত ন্থ ন্দ ন্ধ ন্ন ন্ব ন্ম ন্য ন্ৰ ন্স ন্ত্ব
ন্ত্ৰ ন্দ্ব ন্থ্ৰ ন্ধ্ৰ
ন্না ন্নি ন্নী ন্নু ন্নূ ন্নৃ ন্নে ন্নৈ ন্নো ন্নৌ ন্নং ন্নঃ
ন্ন্

Figure 3: Third level menu (in red) for a ligature

Thus, in our three-level menu system, we provide direct access to every vowel, vowel diacritic, digit, special punctuation mark, consonant, and consonant-vowel diacritic combination, as well as every ligature-vowel combination. However, this made for an unweildy keyboard with access to over 3000 character combinations in which it is possible to type anything that can appear in print by menu traversal only.

Our tests on this keyboard were disappointing. We were unable to perform a theoretical analysis of this keyboard since we could not find any theoretical models to do so. Our one-time test with 5 volunteers led to a speed of 30 characters per minute or about 5 words per minute. Therefore, we looked for ways to develop more intelligent designs of keyboards. The rest of the paper discusses our new approach.

4 Soft Keyboard Development Using Optimization Techniques

We attempt to address the problems of Brahmic script text input by applying a combination of techniques previously used for the development of English keyboards and those design choices already used for Indic language layouts. We look at two approaches for improving input potential: layouts based on character frequency, and the use of machine learning techniques for character placement.

Keyboard layouts based on alphabetical ordering make the initial learning of the keyboard simpler, however, it is a tradeoff for efficient input later. Based on the improvements gained in English keyboards by organizing layouts by unigraph and bigraph frequency, we theorize that a similar approach will be equally beneficial for Brahmic script keyboards. The Emille Corpus for Assamese (Baker et al. 2003) was analyzed in order to develop tables of unigraph and bigraph frequencies. In addition initial research into ligature frequencies has been performed on the same corpus (Baker et al. 2003). Although ligatures have not yet been integrated into the soft keyboards being developed, future research into this is expected.

4.1 Design Choices

In creating soft keyboards, several design choices were made that may affect input speed. Some of these are discussed here.

In all of the keyboards designed, the spacebar is fixed below the grid of characters. When determining the distance between a digraph that contains a space, the center of the spacebar is used as the location the user would choose when typing. Although optimum input speed is obtained when the user always chooses the shortest path, this cannot be expected. Zhai et al. estimated that, given the choice of four spacebars, users chose the optimum space bar only 38% – 47% of the time (Zhai et al. 2000). Therefore, in order to have a conservative estimate of the upperbound for input time, perfect travel to and from thespacebar was not assumed. In addition, this decision avoids underestimates in movement calculations from “free-warping” in which the stylus enters a spacebar in one location and leaves it in an unrelated location, a common error in soft keyboard evaluation (Zhai et al. 2000).

We have chosen to create rectangular ‘grid’ keyboards in which each character occupies a square. This is the most common layout, however the Metropolis Keyboard (Zhai et al. 2000) and select others use hexagon keys and irregular, honeycomb shapes for the keyboard. Our decision to use a rectangular layout for all of our designs is a desire to have a similar layout among all stages of development in order to best compare them. It is believed that rectangular layered menus will be most efficient because square hierarchical menus have been shown, both theoretically and empirically, to be the most efficient type of menu selection (Ahlström et al. 2010). A desire to have a rectangular keyboard with square keys at one stage necessitates the need for a similar layout and key shape at all stages.

4.2 Genetic Algorithm Designs

Genetic algorithms are a technique in machine learning that is derived from the principles of natural selection. A genetic algorithm is composed of three major parts: a population of potential solutions, a fitness function which evaluates those solutions, and a method for reproducing and changing the population of potential solutions over time. In a given generation, each individual within the current population of potential solutions is evaluated by the fitness function and given a score. The higher an individual’s score, the more fit, or closer to the optimum solution, they are considered to be. A new population of potential solutions is then created, in which the most fit individuals continue to survive and new individuals are created from fit individuals in the previous generation. These new individuals can be a combination of previous individuals, or can be developed by mutating previous individuals. Many iterations of this occurs until the solution, or an approximate for it, is found.

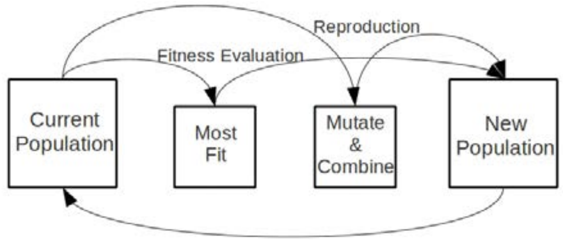


Figure 4: Evolutionary cycle of a population of chromosomes in a genetic algorithm.

In developing a genetic algorithm for soft keyboards, each individual in the population, called a chromosome, is a different layout for a keyboard. Each chromosome contains a number of genes equal to the number of characters in the alphabet being considered. Each gene is unique. In the initial genetic algorithm, a simple one-layer, rectangular keyboard was developed. Each chromosome was given a score in its fitness function that represented the inverse of its mean time to type a character in that layout, as calculated by Fitt's Law. Thus chromosomes with lower mean times were given higher fitness scores. The fastest layouts were kept for the next generation, and chromosomes were chosen for gene mutations (swapping the locations of two characters) and combinations with other chromosomes (in which part of one layout was adopted by another layout) with likelihood of being chosen proportional to their fitness. These newly created chromosomes as well as the best performing chromosomes become the population to be evaluated in the next round.

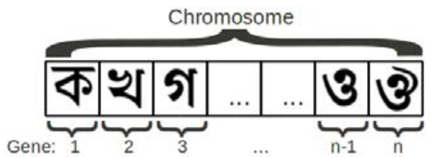


Figure 5: Example chromosome for an Assamese keyboard in which each character in the Assamese alphabet is a gene in the chromosome. In an $a \times b$ rectangular keyboard, the first a genes represent the first row, and the second a genes represent the second row, etc.

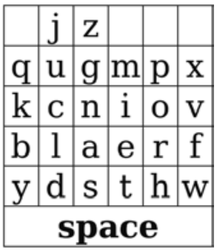
In our genetic algorithms, we allowed the population of keyboards to continue evolving until a single layout was considered the most fit in the population for a predetermined number of continuous generations.

There are many variables in genetic algorithms that can be adjusted, including but not limited to: the number of chromosomes in a population, the number of generations the genetic algorithm runs for, the number of required stable generations (one chromosome is consistently the most fit), the chance of mutation, and how many chromosomes were preserved between generations. There are too many variables to fully test all possibilities. When determining what values to use for the chance of mutation, we tested a range of percentages between .001 and .15 and found our best results with a chance of approximately .08. This is a higher mutation rate than many genetic algorithms use, and further testing may show that better or faster results are found with a different mutation rate. However, we used mutation rates of .08 for most of our tests whose results are reported in this paper. We also chose to preserve approximately 10% of each population.

We chose to focus our attention on varying the number of chromosomes in a population and the number of required stable generations. For each keyboard designed using genetic algorithms we varied the population size between 10 chromosomes and 1000 chromosmes. Although the results varied, generally it seems that the best performing keyboards were developed with larger population sizes. The number of required stable generations were varied between 10 and 100. A couple of tests were run with a stable generation requirement of 250 and 500 and the results were not any better. Having a high number of stable generations didn't seem to improve the results. Most of the best keyboards had a stable generation requirement of 15 to 35. Although many tests were run for each type of keyboard developed, only the best keyboards from each category are reported in this paper; we note the number of chromosomes used to develop that keyboard as well as the number of stable generations required.

5 GA-Based Flat English Soft Keyboards

The keyboards previously designed for Brahmic scripts have two downfalls as comparators for our keyboards: they have only been evaluated by human volunteers, and there has been little attempt to optimize them. As a result, in order to determine that the techniques we use to design keyboards are indeed designing theoretically efficient layouts, English keyboards were designed and evaluated in parallel with Brahmic keyboards. This allowed us to see whether the techniques create keyboards that are competitive in a language where much research for optimization has taken place. If these keyboards are theoretically competitive in English, we hypothesize that the same techniques applied to the development of Brahmic keyboards will also create keyboards that will be competitive in terms of efficiency even though no previous work on theoretic analysis of input speed has previously been performed.



	j	z			
q	u	g	m	p	x
k	c	n	i	o	v
b	l	a	e	r	f
y	d	s	t	h	w
space					

Figure 6: English soft keyboard developed using genetic algorithms. This keyboard has a predicted upper-bound of 38.6 wpm based on Fitt's law. While this is not as high as other upper-bounds, the suspected difference is the placement of the spacebar across the bottom of the keyboard rather than including it as a gene in the chromosomes in the genetic algorithm.

5.1 Fitness Function Used: Fitt's Law

Fitt's Law (Fitt 1992) is a technique used for evaluating a theoretic upper-bound of words per minute (wmp) in stylus-based input systems. While Fitt's Law has been widely used for evaluation of soft keyboards in English, it has not previously been applied to Brahmic script keyboards. It calculates the mean time in seconds to type a character, which can then be used to determine wpm (Zhai et al. 2000).

Fitt's Law finds the average time to move between a character i and a character j in an alphabet with n characters by looking at the distance apart the characters are, D_{ij}, as well as the frequency with which that digraph occurs, P_{ij}. Given a width W_j for each key j, and an index of performance IP, the mean time in seconds to type a character is:

$$\bar{t} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{P_{ij}}{IP}\left[log_2(\frac{D_{ij}}{W_j}+1)\right]$$

Previous work in the field uses an IP of 4.9 (MacKenzie and Zhang 1999) (Raynal and Vigouroux 2005) (Zhai et al. 2000). In order to maintain consistency in evaluation, our calculations also use this value.

In the tests performed on the designed English keyboards, the standard of five characters per word is used in our computations. Thus, given the mean time in seconds, $\bar{t}$, for typing a character, the calculation for wpm is $wpm = \frac{60}{5\bar{t}}$.

The fitness function used to develop both English and Assamese flat keyboards was an inverse of Fitt's Law. In order to maximize the input speed of the user, our fitness function sought to minimize the mean time to travel between characters. In other words, the fitness function evaluated each layout accoriding to Fitt's Law. The higher the Fitt's score the keyboard received, the greater the mean time to travel between characters, and the lower the fitness score given to the keyboard. The higher the fitness score of a keyboard, the more likely it was to be kept for the next generation and to be selected for crossover and mutation.

5.2 Evaluation

The only soft keyboards developed by us for English that we have tested were done using a genetic algorithm that employed Fitt's Law as the fitness function. These keyboards were designed to parallel the most basic Brahmic keyboard in which only one layer is used. While our designed English keyboard is not the fastest according to Fitt's Law, it is competitive with other keyboards. Although its theoretic upper-bound is less than 40 wpm, as compared to 43.1 wpm (Zhai et al. 2000) and 46.4 wmp (Raynal and Vigouroux 2005), there is a considerable enhancement over using a standard QWERTY layout as a soft keyboard. We hypothesize that the reason our keyboard is slower is that we used a fixed space-bar at the bottom of the keyboard rather a central button (Zhai et al. 2000) or including multiple space buttons (Raynal and Vigouroux 2005). The result is an increase in the average travel distance to and from the space-bar, the most frequently occurring character. A test with human volunteers was carried out to show that the typing trend was approaching the expected words per minute as predicted by Fitt's Law. Ten volunteers typed random phrases on the keyboard during ten minute sessions over a period of a week. The phrases used were from MacKenzie and Soukoreff's published collection (MacKenzie et al. 2003). This test was not intended to determine input speed of long-term users, but to determine if the keyboard is easily learned and to show that input speed does approach the predicted speed. The results suggest that user's typing speed does increase and will continue increasing after long term use. Therefore, we claim that the genetic algorithm used to design this keyboard, as well as using Fitt's law to evaluate it, are both valid for developing and evaluating soft keyboards, and we will apply them to Brahmic scripts as well.

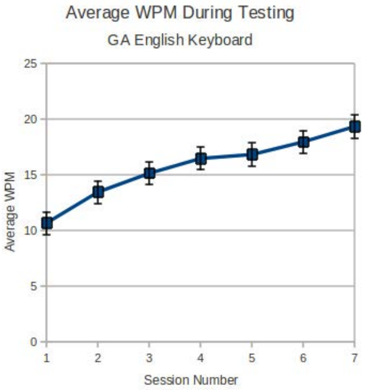


Figure 7: Graph of volunteers' average improvement over multiple ten-minute sessions at typing on the English keyboard designed using genetic algorithms. Average user input speed is increasing with each session.

6 GA-based Flat Brahmic Keyboard

As a basis to compare the results of our genetically designed keyboards to, we first evaluated a flat, rectangular, alphabetic keyboard. In order to maintain the comparison, the diacritics were also added to the keyboard. Several layouts, which varied in number of columns and rows, were evaluated using Fitt's Law. In addition, the placement of the diacritics was adjusted. In all trials, the vowels and consonants were kept in alphabetic order and the diacritics were kept in a group, but the input speed was calculated with three different layouts: with the diacritics appearing before the vowels, after the vowels, and after the consonants. The fastest resulting alphabetic keyboard, shown in Figure 8, was an 8 × 8 square with the diacritics appearing after the consonants. An evaluation using Fitt's Law predicts an input speed of 25.06 wpm.

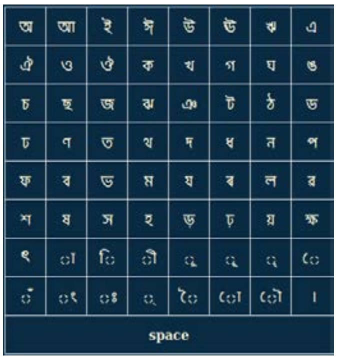


Figure 8: Alphabetically Organized Keyboard for Assamese with diacritics located after consonants. This keyboard is the predicted fastest alphabetic keyboard we tested and has a theoretic input of 25.06 wpm.

The first Brahmic script keyboard designed using genetic algorithms was a simple, single-layer keyboard for Assamese. The keyboard maintained the 8 × 8 grid of the 64 consonants, vowels, and diacritics that was used in analyzing an alphabetically organized keyboard. It is an improvement on the alphabetic keyboard, as its design considers the bigraph frequencies of Assamese and attempts to minimize the travel distance between characters in frequently occurring bigraphs. It was designed using a population of 1000 layouts which were allowed to evolve until a single keyboard had been the "most fit" for 25 generations. Although many tests were performed with varying numbers for population size and required stable generations, this was the best performing keyboard we found. The resulting keyboard has an expected input of 34.23 wpm according to Fitt's law, a promising initial result. When designing GAG1 and GAG2, Raynal and Vigouroux used populations of 20, 000 chromosomes and continued their algorithm until a single keyboard had been the most fit for 500 generations (Raynal and Vigouroux 2005).

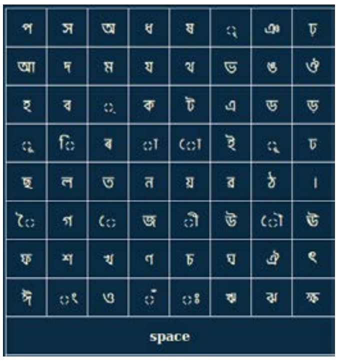


Figure 9: Assamese soft keyboard developed using genetic algorithms. This keyboard has a predicted upper-bound of 34.23 wpm based on Fitt's law.

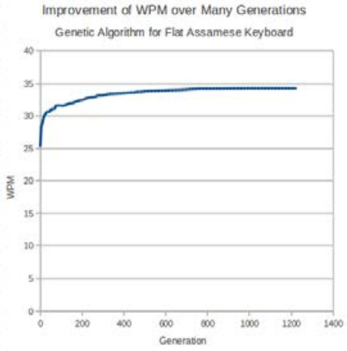


Figure 10: A graph of the growth of the improvement of the most fit keyboard in the genetic algorithm. There were 1000 keyboard layouts in each generation and the algorithm continued until the population had been stable for 25 generations.

In determining words per minute for Assamese soft keyboards, several measurements had to be obtained. The Emille Corpus for Assamese was used to calculate frequencies for all unigraphs and bigraphs. In addition, it was used to obtain the average number of characters per word in Assamese, which was calculated to be approximately six (Baker et al. 2003).

7 Hierarchical GA-Based Brahmic Keyboard

The initial use of genetic algorithms for designing soft keyboards for Brahmic scripts based on character frequency is very promising. A second technique tried for improving input speed is the development of multi-layer keyboards. A multi-layered keyboard allows menus or extra keys to be placed on top of an original keyboard. Genetic algorithms have previously been shown to produce high efficiency in the development of hierarchical menus (Matsui and Yamada 2008). While hierarchical menus and layered keyboards have some differences in design, the techniques used to evaluate them will be similar. For example, a two layer keyboard in which the second layer is accessible by rolling over characters in the first layer can be represented by a two-level menu.

7.1 Diacritic Menu as Second Layer

In an attempt to reduce the number of keys on the screen as well as improving input speed, the technique of adding a diacritic menu to each consonant was tried, When a consonant on the main level of the keyboard is selected by pressing the mouse button down, a diacritic menu appears around the selected consonant. The user can either release the mouse button on the chosen consonant to type a bare consonant, or they can drag the mouse and release it above one of the diacritics to add it to the consonant.

The diacritic menu allows selection of the 12 most frequently occurring diacritics. They were hand-placed in the diacritic menu such that the four most frequent diacritics appear immediately above, below, and to the side of the selected consonant. The next four most frequently occurring diacritics appear in the location immediately diagonal from the selected consonant. This is intended to allow the fastest possible selection time. Future work may include using optilization techniques to place the vowels, rather than hand-placing them.

The diacritics appear in the same location

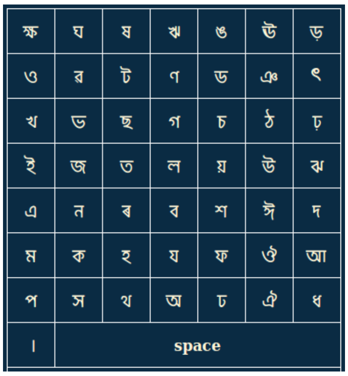


Figure 11: A layered keyboard for Assamese. Consonants and vowels are organized on the bottom layer using a genetic algorithm.

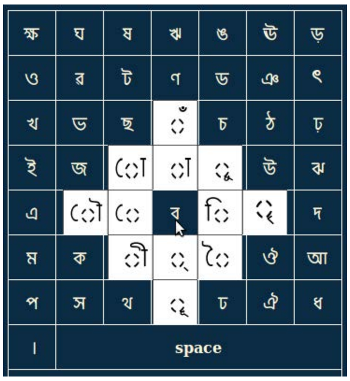


Figure 12: The second layer appears when a consonant is pressed down. The most common diacritics are available to be added to the consonant. They always appear in the same location in the diacritic menu.

for each consonant regardless of frequency with which they are used together. This is anticipated to facilitate faster learning of the diacritic menu by minimizing the visual search time for each consonant and decreasing the number of locations that must be memorized by the beginner in order to become an expert.

Diacritic sub-menus are not new, they have been widely used to decrease the number of visible keys. However, to improve upon this idea, we combined the ideas of diacritic menus and the organization by frequency of a base layer of consonants and vowels. We designed a genetic algorithm that took into account the diacritic layer with its fixed diacritic locations within that layer. The idea behind the algorithm is that if there is a frequently occurring trigraph *abc* with a diacritic as the central character, the best location for the third character, *c*, is as close as possible to directly below the diacritic *b* that appears in the menu off of *a*.

Our genetic algorithm was run with a population of 500 individuals with a stable generation requirement of 15. The resulting keyboard, seen in Figure 12, has an expected theoretic input speed of 40.24 wpm. This is a significant improvement over the single-layer, alphabetically organized Assamese keyboard.

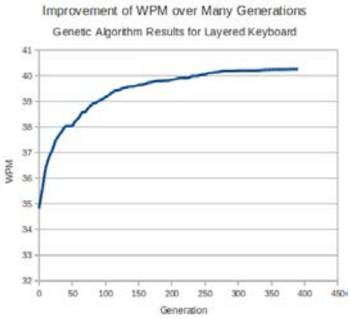


Figure 13: A graph of the growth of the improvement of the most fit keyboard in the genetic algorithm. There were 500 keyboard layouts in each generation and the algorithm continued until the population had been stable for 15 generations.

For comparison, a diacritic layer was added to the alphabetic keyboard. The resulting keyboard had an expected input of 33.94 wpm.

7.2 Vowel Menu

Assamese vowels are used infrequently, the most frequent character occuring only 1.36% of the time. Because of this, we hypothesized that the input speed could be increased by placing all of the vowels in a separate vowel menu. A single button that includes all of the vowels is a common organization technique in previous Brahmic keyboards. This technique has the benefit of reducing the number of characters on the board to be memorized as well as allowing the remaining characters to be closer together. Immediately, it seems that it would improve the theoretic input speed according to both Fitt's Law and Hick-Hyman's law. To test this, the vowels were placed in set positions, based on frequency just like the diacritic menu, in their own menu on the bottom left of the screen, shown in Figure 14.

A genetic algorithm was designed and run to create a keyboard with only consonants in the base layer, a second layer of diacritics attached to each consonant, and a vowel menu button.

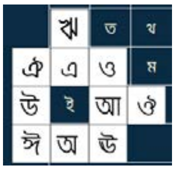


Figure 14: Vowel menu appears around the most commonly used vowel. The vowel menu was located in the lower left of the keyboard.

The keyboards designed using this algorithm were evaluated with the same methods used to evaluate the diacritic menu. However, the best keyboards developed using the vowel menu had, on average, an expected input rate that was 5 wpm less than those expected for the keyboards that had no vowel menu. The keyboard with the fastest expected input rate that included a vowel menu was 35.6 wpm. While further research into different layouts of the vowel menu and different locations for it may create a keyboard that has an input expectation similar to those without a vowel menu, 5 wpm is a significant gap to bridge.

7.3 Fitness Function Used

While an inversion of Fitt's Law was an effective fitness function for a single-level soft keyboard, multi-level keyboards need a different one. Multiple layers means that traversal time between layers needs to be taken into account, both in evaluation and in determining the most fit keyboards. In order to do this, we applied the ideas from the evaluation of hierarchical menus performed by (Ahlström et al. 2010). They built on a technique for analyzing the search and selection time of items in a hierarchical menu from (Cockburn et al. 2007) that combines Fitt's Law, Hick-Hyman's Law, and the Steering Law and applied it to hierarchical menus in which each menu and submenu is a square grid of the available options. The rectangular, multi-level soft keyboards we designed are an application of a rectangular hierarchical menu, and so their techniques for evaluation were used in the fitness function of our genetic algorithms (Cockburn et al. 2007).

Hick-Hyman's Law (Seow 2005) predicts the amount of time a person will take to make a decision given the number of choices they have. The user's reaction time is modelled by

$$T = b\sum_{i=1}^{n} p_i \log_2(\frac{1}{p_i} + 1)$$

where n is the number of choices and p_i is the probability of that choice being chosen. A user must make a decision about what character to press when their selection changes. The Hick-Hyman law can be applied to the original, base-layer of the keyboard as well as to the diacritic second layer.

The Steering Law claims that the time to move the cursor through a constrained two-dimensional "tunnel" to move from one level to the next is a linear function of the ration between the tunnel length and width. The Steering Law is very important in classic hierarchical menus and pie menus. When the hierarchical menu is a square grid, however, the ratio reduces to one and the time to move into a new level approaches the predicted time to move between two points as predicted by Fitt's Law (Cockburn et al. 2007). The diacritic menu that appears around the selected consonant is mostly square, but includes some outcroppings. The Steering Law does not need to be taken into account when selecting consonants and vowels on the base-layer.

The fitness function used in the our development of Assamese soft keyboards with layers attempts to minimize the sum of the time spent searching for the next character (Hick-Hyman's Law), moving between locations (Fitt's Law) and moving between layers (The Steering Law). The fitness function gets the mean time to type a character by summing three different "types" of ways a character can be typed. There are two types of characters that can be typed. Base characters reside in the bottom layer of the keyboard and diacritics appear in the second level. The three types of inputs that must be taken into account are a base character typed after another base character, $\bar{t}_{bb}$, a base character followed by a diacritic character, $\bar{t}_{bd}$, and a diacritic character followed by a base character, $\bar{t}_{db}$.

$$\bar{t} = \bar{t}_{bb} + \bar{t}_{bd} + \bar{t}_{db}$$

The goal of the fitness function, as before when the average time was computed only with $\bar{t}_{bb}$, is to reward chromosomes that have smaller $\bar{t}$ values with higher fitness scores.

In calculating $\bar{t}$, the three averages are computed as follows. $\bar{t}_{bb}$ is the sum of Fitt's Law and Hick-Hyman's Law for each pair of characters on the base layer. Steering's Law does not apply to the base layer because the ratio between the width and height of the "tunnel" being traversed is 1. $\bar{t}_{bd}$ is the sum of Hick-Hyman's Law and the Steering Law for each diacritic that can be chosen from each base character. $\bar{t}_{db}$ is more complicated because it requires calculating the distance from each diacritic to each base character, and all possible locations of each diacritic must be taken into account. This requires trigraph analysis of each base-diacritic-base combination. $\bar{t}_{db}$ is the sum of Hick's Law and Fitt's Law for all base-diacritic-base trigrams.

7.4 Implementation of Text Prediction

Text prediction has been shown to help improve input speed in several input domains such as texting and as an input aid for persons with disabilities (Wobbrock and Myers 2006). Although Brahmic script input does not fall under those domains, it has been hypothesized that well designed and implemented text prediction could increase the input speed of the average computer user (Anson et al. 2006). Although text prediction has not caught on as a general typing aid because it requires that the user switch their focus from the keyboard on which they're typing to the prediction suggestions, this may not be as large of a problem when typing with soft keyboards. The study performed by Anson et al. suggests that text prediction algorithms used in conjunction with soft keyboards actually improve the input speed. They hypothesize that this is because the predictions are physically closer to where the typist is focused, and so requires less time to use (Anson et al. 2006). The result of this study motivates us to incorporate text prediction into our Brahmic keyboards.

Unfortunately, there has been little theoretic work on the time required to consider the predictions suggested. Since the typist will glance at the list of suggestions after every character they type regardless of whether the word they are writing appears there, it is apparent that poorly designed and implemented text prediction could easily decrease input speed. Empirical tests have shown that vertically listed suggestions tend to be faster, more suggestions increases the input time, and that using n-grams helps with prediction suggestion when n, the number on words used, is not too large (Garay-Vitoria and Abascal 2005).

Our implementation of text prediction uses unigram, single word, and n-gram, multi-word, prediction. Once a user has begun typing a word, the three most frequently occuring words beginning with the letter the user has typed are displayed for them to choose from. If the user does not select one of those words but instead types a second character, the most frequently occuring unigrams beginning with those letters are suggested. This continues until the user has either typed the full word or selected a word from the suggestion list. When a word has been completed by either method, three words are suggested as the next word. Next word predictions are made by using bigrams and trigrams, of which the next word is the last word in said n-grams. The three most frequently occuring trigrams using the two most recently typed words as the first and second word in the trigram are suggested. If there are no trigrams, then bigram prediction, with the most recently typed word as the first word in the bigram, is used instead. Using an n-gram model is hoped to improve accuracy of predictions.

The frequencies for both unigrams, bigrams, and trigrams are based on our analysis of the Emille corpus for Assamese (Baker et al. 2003). Our predictions are listed vertically on both sides of the keyboard, as can be seen in Figure 15. The top word has the highest frequency and the bottom word has the lowest frequncy of those listed. Locating the suggestions on both sides of the keyboard is hoped to minimize gaze and travel time for the typist, as they can use whichever prediction list is closest to the last character they typed.

Our current work regarding text prediction lies in the realm of using the previously typed words in a document to predict the

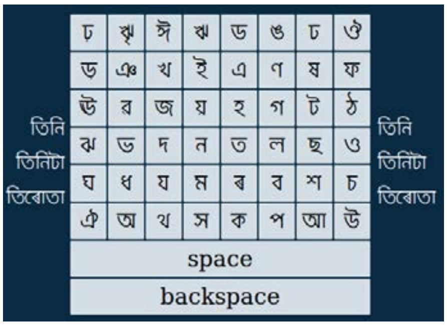


Figure 15: Text prediction suggests three words beginning with the letters already typed that have the highest frequency of occurence among all unigrams. The suggests appear on both sides of the keyboard.

words that will be typed. Typists tend to reuse the same vocabulary, at a rate of approximately 70% (Tanaka-Ishii et al. 2003). Incorporating the frequency of use of certain words by a particular user would make the word suggestions more accurate. Many previous solutions to the problem of user-affected frequencies have used a personalized dictionary that the typist can add words to. However, this is not used as often as it could be because of the inconvenience of adding words. In addition, this technique does not account for a difference in personal vocabulary frequency. A better approach would allow for dynamic frequencies in the text prediction.

8 Future Work

Despite the progress that has been made, there are still many tactics and techniques that could be tried to develop a better keyboard.

Success with genetic algorithms suggests that other optimization techniques can also be applied to soft keyboard development. Such techniques as ant colony algorithms and particle swarm optimization will be investigated and applied to Brahmic soft keyboards.

Additionally, future research into the implementation of gestures, in which a certain movement with the stylus or mouse indicates a different level, will be investigated. Using gestures as a second or third layer above the original keyboard may allow for easier and faster vowel diacritic selection. It is expected that ultimately the best keyboards will be a combination of these tactics.

Another technique that may improve input speed is the inclusion of ligatures. Ligatures take more time to input than other characters because a single ligature requires three or five characters to be selected on the keyboard for a single character to be displayed in the output. Inclusion of a few commonly used ligatures may increase input time by decreasing the number of key strokes that must be made.

Perhaps the most important step in this research will be empirical testing. Tests with volunteers on the layered and predictive Assamese keyboards will be performed. Testing the Assamese keyboards will reveal whether they are easy to learn and use by people. Although they may be theoretically efficient, keyboards are not worthwhile if they're not usable. In addition to the benefits of empirical testing on the genetically engineered Assamese keyboards, tests will provide a basis to compare the results of testing the keyboards that employ text-prediction. Because we have not yet found a way to theoretically analyze the input speed of keyboards with text prediction, empirical testing is essential. Such a test would compare the input speed of volunteers on one of the previously mentioned keyboards as well as their speed on the same keyboard that includes text prediction.

Assamese is not the only language that could benefit from an optimized keyboard. We are looking into developing keyboards, using the same techniques as used with Assamese, for other Brahmic languages such as Gujarati and Kannada. Although many languages share characters, differences in frequency of use of those characters will cause a keyboard optimized for one language to not be very efficient for another language. The benefit of a soft keyboard is that anyone using it would be able to switch the layout to the one they are most familiar with, it is not necessary to have standard character placements for all users in a single county or area.

9 Conclusion

Soft keyboards have the benefit of not being limited to a specific layout as physical keyboards are. They are versatile in shape and can be varied by language in order to be most efficient for whoever is using them. Theoretic analysis using techniques such as Fitt's Law, Hick-Hyman's Law, and Steering's Law were used to analyze various layouts, and showed that reorganizing keyboards such that characters in frequently occuring bigraphs appear near one another results in a keyboard with a higher expected input rate.

Table 1: Summary of Expected Input Speeds

Keyboard Type	Expected WPM
Flat Alphabetic	25.06
Layered Alphabetic	33.94
Flat GA-Designed	34.23
Layered GA-Designed	40.24

Assamese text input can be improved from an expected input of approximately 25 wpm using an alphabetically ordered keyboard to approximately 34 wpm merely by reordering the keys. This improvement has also been seen in English, and can likely be applied to other Brahmic scripts. In addition, applying this technique of organization by frequency using through genetic algorithms to layered keyboards can lead to another significant increase in expected input to approximately 40 wpm. It is interesting to note that alphabetic keyboards that apply layering techniques are still outperformed by a flat keyboard organized by frequencies. This suggests that the most important technique for improving input speed is frequency organization. This would mean that having efficient keyboards designed for each language is more important than a general keyboard design for a shared script. The beauty of soft keyboards is that they allow this versatility. Once efficient keyboards have been developed for a language, it can be available for speakers of that language everywhere they go.

References

Ahlström D., Cockburn A., et al., *Why it's Quick to be Square: Modelling New and Existing Hierarchical Menu Designs.* Dallas, PA, USA: College Misericordia, 2010.

Anson D., Moist P., et al., *The Effects of Word Completion and Word Prediction on Typing Rates Using On-Screen Keyboards.* Atlanta, GA, USA: CHI 2006.

Baker P., Monmarche N. et al., *EMILLE Corpus: Assamese.* European Language Resources Association, 2003.

Cockburn A., Gutwin C., and Greenbrug S., *A Predictive Model of Menu Performance.* San Jose, CA, USA: CHI, 2007.

Colas S., Hardy A. et al., *Artificial Ants for the Optimization of Virtual Keyboard Arrangement for Disabled People.* Tours, France: Laboratoire d'Informatique de l'Universite de Tours, 2008.

Columas, F., *The Writing Systems of the World*, Cambridge, MA: Basil Blackwell, 1990.

Fitts, P. M., *The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement*, Journal of Psychology: General, Volume 121, No. 3, pp. 262-269 (reprint of the same article from Journal of Experimental Psychology, 47, 381-391, 1954), 1992.

Garay-Vitoria N., Abascal J., *Text Prediction Systems: A Survey.* Donostia, Spain: KReSIT IIT Bombay, 2005.

Joshi A., Parmar V. et al., *Keylekh: A Keyboard for Text Entry in Indic Scripts.* Mumbai, India: KReSIT IIT Bombay, 2004.

Krishna A., Ajmera R., Halarnkar S. and Pandit P., *Gesture Keyboard - User Centered Design of a Unique Input Device for Indic Scripts.* Mumbai, India: HP Laboratories, 2005.

MacKenzie S. and Soukoreff R. W., *Phrase Sets for Evaluating Text Entry Techniques.* Ft. Lauderdale, FL, USA, 2003. 1992.

MacKenzie S. and Zhang S. X., *The Design and Evaluation of a High-Performance Soft Keyboard.* Pittsburgh, PA, USA, 1999.

Matsui S. and Yamada S., *Genetic Algorithm Can Optimize Hierarchical Menus.* Florence, Italy, 2008.

Rathod A. and Joshi A., *A Dynamic Text Input Scheme for Phonetic Scripts like Devangari.* Mumbai, India: IIT Bombay, 2002.

Raynal M. and Vigouroux N., *Genetic Algorithm to Generate Optimized Soft Keyboard.* Portland, OR, USA, 2005.

Seow, S., *Information Theoretic Models of HCI: A Comparison of the Hick-Hyman Law and Fitts' Law*, Human-Computer Interaction, Volume 20, pp. 315-352, 2005.

Shanbhag S., Rao D., and Joshi R. K., *An Intelligent Multi-Layer Input Scheme for Phonetic Scripts.* 1em plus 0.5em minus 0.4emMumbai, India, 2002.

Sowmya V. B. and Varma V., *Design and Evaluation of Soft Keyboards for Teluga.* Pune, India, 2009.

Tanaka-Ishii K. et al., *Acquiring Vocabulary for Predictive Text Entry through Dynamic Reuse of a Small Corpus.* Tokyo, Japan: University of Tokyo, 2003.

Venolia D., and Neiberg F., *T-Cube: A Fast, Self-Disclosing Pen-Based Alphabet.* Boston, MA, USA: Human Factors in Computing Systems, 1994.

Wobbrock J. O., and Myers B. A., *From Letters to Words: Eficient Stroke-based Word Ccompletion for Trackball Text Entry.* Portland, OR, USA: ASSETS, 2006.

Zhai S., Hunter M., and Smith B. A., *The Metropolis Keyboard - An Exploration of quantitative Techniques for Virtual Keyboard Design.* San Diego, CA, USA: IBM Research Center, 2000.

Multiple Post Microblog Summarization

David Inouye
University of Colorado at Colorado Springs

***Abstract*—The use of microblogs such as Twitter[1] has increased incredibly over the past few years. Because of the public nature and sheer volume of text from these constantly changing microblogs, it is often difficult to fully understand what is being said about various topics. A method for summarizing popular topics of microblogs has been proposed but its summaries are only one sentence or phrase in length. Therefore, this work focuses on extending microblog summarization by producing multiple post summaries. Two main summarization algorithms are explored: a clustering based algorithm and a threshold based Hybrid TF-IDF algorithm. The results will be evaluated by comparing the generated summaries with manually generated summaries. For purposes of comparison, the results are also compared to MEAD, LexRank and TextRank—some leading traditional multi-document summarization systems.**

***Index Terms*—microblogs, summarization, clustering.**

I. Introduction

THE massive rise of microblogging as a new form of communication and data generation[2] has opened up a new area of natural language processing that could be aimed at discovering real time public opinion or news stories. In order to understand and use this amount of information, however, automatic summarization is a necessity. Though automatic summarization of longer or more structured documents has been researched [1]–[9], processing short and unstructured microblog posts has only recently been considered. Sharifi, Hutton and Kalita [10] proposed and implemented two methods of summarizing any given microblog topic in one sentence.[3] They implemented a novel graph-based Phrase Reinforcement algorithm and a Hybrid TF-IDF algorithm. The Hybrid TF-IDF algorithm was an altered TF-IDF algorithm (explained in [11], [12]) in which the TF (term frequency) component is computed upon the entire collection of posts whereas the IDF (inverse document frequency) component is computed upon a single post. Both algorithms produced human competitive results but the Hybrid TF-IDF algorithm seemed to produce consistently better results and therefore is integrated into this project.

II. Motivation

Though microblog summarization algorithms have produced summaries competitive with human generated summaries [10], they only produce a single sentence. Consequently, they can only represent one idea surrounding a topic. With this limited coverage of a specified microblog topic, important or interesting information about a topic may be easily overlooked. Though these short summaries may provide simple *indicative* summaries that give enough information to spark the interests of users as explained in [5], multiple post summaries that cover multiple subtopics of the original topic would push the summaries towards being *informative* [5]. Therefore, this paper describes some possible methods for producing these multiple post summaries for microblogs.

III. Problem Definition

The problem considered in this paper is how to produce a multiple post summary of microblog posts on a particular topic that is specified by a keyword or phrase. The resulting summary will be an extractive summary because the algorithms presented in this paper extract quotes from the original collection of posts.

This problem can be defined as follows: given a topic keyword or phrase T and the number of posts for the summary k, retrieve a set of microblog posts P in which for all $p_i \in P$, T is in the text of p_i and output a set of posts S with a cardinality of k in which all $s_i \in S$ are related to T by a relative relevancy ranking r_i but for all $s_i, s_j \in S, s_i \not\sim s_j$. For each post $s_i \in S$, a feature vector $v_i \in V$ can be computed based on word frequencies after any noise has been removed. The primary measure of similarity will be the cosine similarity measure:

$$\mathrm{sim}(s_i, s_j) = \cos(v_i, v_j) = \frac{v_i^t v_j}{\|v_i\|\|v_j\|},$$

which can be simplified to $\mathrm{sim}(s_i, s_j) = v_i^t v_j$ because $\|v_i\| = \|v_j\| = 1$.

IV. Proposed Solution

Two main methods for producing summaries are explored in this paper. The first method is using clustering to cluster posts into subtopics and then summarizing each cluster individually and the second method is modifying the Hybrid TF-IDF summarization algorithm so that it can produce multiple post summaries. Initial testing of microblog post clustering are performed to select the best clustering method for the cluster summarier.

A. Clustering Microblog Posts

In this phase, Sharifi's program filters the posts by removing any non-English posts and spam messages as determined by simple heuristics and a spam classifier. Then, post noise such

D. Inouye is participating in a Research Experience for Undergraduates (REU) with the Department of Computer Science, University of Colorado, Colorado Springs, GA, 80918 USA.

[1] http://twitter.com

[2] http://www.networkworld.com/news/2010/041410-biz-stone-says-twitter-has.html

[3] The program developed by [10] will be referred to as "Sharifi's program" throughout the rest of the paper.

as html tags, website addresses, headings and references are removed.

Once the posts have been pre-processed, the feature vectors $v_i \in V$ will be computed for each post based on the Hybrid TF-IDF weighting of words already implemented in Sharifi's program. In order to increase the performance of the algorithms, the feature vector computation ignores two types of noise: stop words that appear in a large majority of posts such as "a," "and" or "the" and simple sentences in a post such as "Wow!!!" and "Hahaha, that's funny." These feature vectors are normalized to unit vectors so that $v'_i = \frac{v_i}{||v_i||}$ in order to account for the difference in post lengths [13]. However, for comparision, the first set of tests does not normalize the feature vectors. The posts are then processed by a variety of greedy clustering algorithms. The main set of algorithms are variations of the k-means algorithm. Because the standard k-means algorithm generally performs well and is easy to implement [14], it has been tested first. The bisecting k-means algorithm was implemented afterwards because it may perform better than the direct k-means algorithm as suggested in [13], [14]. The k-means++ algorithm, which is a new variation of k-means algorithm proposed and initially tested by Arthur [15], was then implemented. Finally, an algorithm that combines the k-means++ algorithm with the bisecting algorithm was implemented.

For the following definitions, the centroid $c_i \in C$ is defined as

$$c_i = \frac{\sum_{v \in V_i} v}{n_i},$$

in which n_i is the number of posts and V_i is the set of all feature vectors in ith cluster. The k-means clustering algorithms are defined as follows:

1) Standard k-means algorithm
   a) Randomly choose k initial centroids $c_i \in V$ from all the computed feature vectors.
   b) Assign each post p_i to the centroid that is most similar to its corresponding feature vector v_i.
   c) Compute the centroid of each cluster.
   d) Repeat steps 1b and 1c until no posts are reassigned.
2) Bisecting k-means algorithm
   a) Split the set of posts P into 2 clusters using the standard k-means algorithm ($k' = 2$) defined by step 1 above.
   b) Choose an already formed cluster to split.
   c) Repeat steps 2a and 2b until the kth cluster has been formed.
3) k-means++ algorithm
   a′) Choose initial centroids based on probability.
      i) Choose an initial centroid c_1 uniformly at random from V.
      ii) Choose the next center c_i, selecting $c_i = v' \in V$ with the probability $\frac{D(v')^2}{\sum_{v \in V} D(v)^2}$ where $D(v)$ is the shortest distance from v to the closest center which is already known.
      iii) Repeat step 3a′ii until k initial centroids have been chosen.
   b-d) Continue with the standard k-means clustering algorithm defined in steps 1b-1d.
4) Bisecting k-means++ algorithm
   a) Follow step 2a of the bisecting algorithm above except use the k-means++ algorithm instead of using the standard k-means algorithm.
   b-c) Continue with the bisecting k-means clustering algorithm defined in steps 2b-2c.

B. Summarization

1) Baseline Summarizers:

a) Random Summarizer - For a baseline, a random summarizer was implemented that randomly chose four posts out of all the posts in each topic to serve as a summary.

b) Mead Summarizer - For the well-known well known multi-document summarization system called MEAD[4] described in [16], the summaries were summarized with the default settings.

c) LexRank Summarizer - LexRank [17] is a graph based multi document summarization method that uses the similarity between two sentences as the weight of the edge between those two sentences. Then, the final score of a sentence is computed based on the weights of the edges that are connected to it. Since the MEAD summarization toolkit came with a LexRank feature script, the LexRank implementation with the MEAD toolkit was used to compute the LexRank summaries.

d) TextRank Summarizer - TextRank [18] is another graph based method that comes primarily from the ideas behind the PageRank [19] algorithm. Because the exact implementation was not available, the summarizer was implemented internally using the formulas described in [18].

2) Cluster Summarizer: The cluster summarizer used the best clustering algorithm found in the clustering tests—the normalized bisecting k-means++ algorithm. It clustered the posts into 4 clusters (k = 4) and then summarized each cluster with the Hybrid TF-IDF algorithm.

3) Variable Cluster Summarizer: This test varied the value of k for k-way clustering from 5 to 10 in order to see if changing the number of clusters affected the results of the Cluster Summarizer. The largest 4 clusters were then chosen, and then each of these four clusters was summarized with the Hybrid TF-IDF algorithm.

4) Hybrid TF-IDF Summarizer: This algorithm developed by [10] weights all the sentences based on a modified TF-IDF (Term Frequency Inverse Document Frequency) weighting of sentences. The definition of what a document is for microblog posts needed to be modified. Therefore, a hybrid definition of a document was used instead in which the TF component uses the entire collection of posts as one document while the IDF component is computed on each post individually. Originally, the algorithm only selected the best summarizing topic sentence, but for this project, it was modified to select the top four most weighted posts.

[4] http://www.summarization.com/mead/

V. Experimental Setup

A. Evaluation Methods

In order to test the clustering algorithms, a testing corpus of pre-classified posts has been fed to the algorithm and the values of entropy and purity has been used as the primary metrics as defined by [13]. Given a particular cluster X_i of size n_r, the entropy of the cluster is defined as

$$E(X_r) = -\frac{1}{\log q}\sum_{i=1}^{q}\log\frac{n_r^i}{n_r},$$

where q is the number of classes in the pre-classified posts and n_r^i is the number of posts of the ith class that were assigned to the rth cluster. The total entropy of the clustering solution is

$$E(X) = \sum_{r=1}^{k}\frac{n_r}{n}E(X_r).$$

In general, the smaller the entropy values the better the clustering solution. Similarly, the purity of a particular cluster is defined as

$$P(X_r) = \frac{1}{n_r}\max(n_r^i),$$

which represents the fraction of the cluster that is made up of the largest class of documents. The total purity of the clustering solution is

$$P(X) = \sum_{r=1}^{k}\frac{n_r}{n}P(X_r).$$

In general, the larger the purity values, the better the clustering solution.

In order to test the final automatic summaries, the ROUGE-N metric [20] will be used to compare manually generated summaries to the automated summaries because [10] found that for microblogs the ROUGE-1 (N = 1 for unigrams) metric is a sufficient evaluation of microblog summaries. Given that M is the set of manual summaries and u is the set of unigrams in a particular manual summary, ROUGE-1 can be defined as

$$\text{ROUGE-1} = \frac{\sum_{m\in M}\sum_{u\in m}\text{match}(u)}{\sum_{m\in M}\sum_{u\in m}\text{count}(u)},$$

where count(u) is the number of unigrams in the manual summary and match(u) is the number of co-occurring unigrams between the manual and automated summaries. This formulation of the ROUGE-N metric can be used to measure the precision of the auto summaries since the divisor is the number of relevant unigrams. The metric can be altered slightly so that it measures the recall of the auto summaries by changing the divisor to be the number of unigrams in the auto summary which represents the number of retrieved unigrams. This formulation of the ROUGE metric can be stated as follows:

$$\text{ROUGE-1} = \frac{\sum_{m\in M}\sum_{u\in m}\text{match}(u)}{M_n * \sum_{u\in a}\text{count}(u)},$$

where M_n is the number of manual summaries and a is the auto summary. Because both recall and precision are important in summaries, the F$_1$-measure of the precision and recall are computed such that

$$\text{F}_\beta - \text{measure} = \frac{(\beta^2+1)pr}{\beta^2(p+r)},$$

where p is the precision and r is the recall. In addition, for reporting the results, the average F$_1$-measure of all the iterations—25 iterations for non-random summarizers and 2500 for random summarizers—was computed as

$$\text{avg(F}_1\text{-measure)} = \frac{1}{F_n}\sum_{f\in F} f,$$

where F is the set of all F$_1$-measures and F_n is the number of F$_1$-measures being averaged.

At least two volunteers will manually generate multiple post summaries by performing all the main steps of the algorithm so that the basic steps parallel the steps of the algorithm. First, like the algorithm, they will cluster the posts into a specified number of clusters k. The specific value of k = 4 was chosen after looking at the posts and determining that on average 4 clusters seemed to be reasonable.[5] Second, they will choose the most representative post from each cluster. And finally, they will order the posts in a way that they think is most logical. The information from this last step was not used in this research project but was collected to possibly help an extension of this project that could deal with the ordering of the posts and post order coherence.

B. Test Data

The test data used in this research project came from the test data collected for Sharifi's summarization program [10]. Over the course of five consecutive days, Sharifi et. al. collected 1500 microblog posts from the top ten trending topics on the Twitter home page. Then, because microblog posts are an unstructured and informal way of communicating, these post were preprocessed to remove spam and other noise features. These pre-processing steps were as follows [10]:

1) Convert any HTML-encoded characters into ASCII.
2) Convert any Unicode characters (e.g. "nu24ff") into their ASCII equivalents and remove.
3) Filter out any embedded URL's (e.g. "http://"), HTML (e.g. "<a.../a>"), headings (e.g. "NEWS:"), references (e.g. "[...]"), tags (e.g. "<...>"), and retweet phrases (e.g. "RT" and "@AccountName").
4) Discard the post if it spam.
5) Discard the post if it is not in English.
6) Discard the post if another post by the same user has already been acquired.
7) Reduce the remaining number of posts by choosing the first 100 posts.
8) Break the post into sentences.
9) Detect the longest sentence that contains the topic phrase.

[5] Though the choice of k will introduce some bias into the manual summary generation, a specific value needs to be set in order to accurately compare the automatic summaries–which take k as a parameter–to the manual summaries especially because the ROUGE-1 metric is very sensitive to summary length. A possible extension of this project would be to design an algorithm to compute the best value for k given a set of posts.

These pre-processing steps and their rationale are described more fully in [10]. Fifty topics of 100 posts gives a total of 5,000 posts. For the clustering tests, the topics were split into 10 sets of 5 so that 5-way clustering could be evaluated over 10 different data sets. For the summarizing tests, only the first 25 topics were used because of the limited number of volunteer hours that were needed to perform manual summaries of the topics.

C. Clustering Test

In order to avoid the sensitivity of random seeding, 100 5-way clustering solutions were computed for each of the 10 different data sets for a total of 1000 iterations per algorithm.

D. Summarization Tests

For the summarizers that involve random seeding (e.g. ClusterSummarizer, RandomSummarizer), 100 summaries were produced for each topic to avoid the effects of random seeding.

For the well-known multi-document summarization system called MEAD[6] described in [16], the posts were summarized with the default settings. Each post was formatted to be one document with a single sentence inside of it.

Since the MEAD summarization toolkit came with a LexRank feature script, the LexRank implementation with the MEAD toolkit was used to compute the LexRank summaries. One change from the main MEAD program test, however, is that all the posts for each topic were added to one document as separate sentences so that their LexRank scores could be computed against each other.

Because the exact implementation of TextRank was unavailable, the summarizer was implemented internally using the formulas described in [18].

The cluster summarizer used the best clustering algorithm found in the clustering tests—the normalized bisecting k-means++ algorithm. It clustered the posts into 4 clusters (k = 4) and then summarized each cluster with the Hybrid TF-IDF algorithm. Because some of the volunteers had suggested that some topics had significant noise and the fact that k-way clustering can perform significantly differently for different values of k, the number of clusters was varied from five to ten and the largest 4 clusters were chosen to summarize.

Because the Hybrid TF-IDF may produce very similar sentences as the top most weighted sentences, a similarity threshold was applied in which the algorithm looped through the posts starting at the most weighted and only choosing the post if the following condition was true for the current post s_i:

$$\text{sim}(s_i, s_j) \leq t$$

for all $s_j \in R$ where R is the set of posts aleady chosen and t is the similarity threshold. The cosine similarity measure was used and the threshold was varied from 0 to 0.99 with increments of 0.01 for a total of 100 tests.

VI. Results and Evaluation

A. Clustering Results and Analysis

he entropy and purity measures of the 1000 iterations for each algorithm were averaged to give an overall sense of each algorithm's performance. The algorithms marked "modified" normalized the feature vectors to unit lengths. The average purities and entropies for all 8 implementations are shown in Table I. In order to give an overview of the relative performance of each implementation, Figure 1 shows the relative entropies and purities that have been normalized based on the following equations:

$$E'_i(X) = \frac{\max(E(X))}{E_i(X)} \text{ and } P'_i(X) = \frac{P_i(X)}{\min(P(X))}.$$

Because of this normalization, higher values are better for both entropy and purity, and all the algorithms are relative to the base k-means algorithm.

TABLE I
Average Entropy and Purities

Algorithm Implementation	Avg. Entropy	Avg. Purity
k-means	0.740	0.491
k-means++	0.731	0.499
Bisecting k-means	0.732	0.504
Bisecting k-means++	0.724	0.509
k-means (modified)	0.732	0.499
k-means++ (modified)	0.724	0.508
Bisecting k-means (modified)	0.720	0.514
Bisecting k-means++ (modified)	0.709	0.525

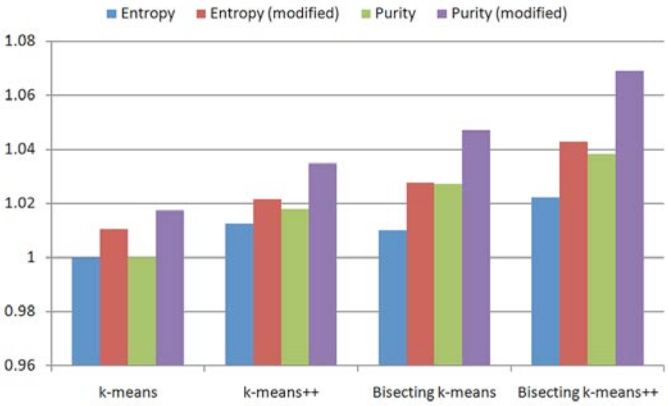


Fig. 1. The normalized relative entropies and purities for each algorithm. The series labeled "modified" use the normalized unit feature vector.

One of the first observations is that normalizing the feature vectors to be unit vectors improved the performance of all the algorithms by about 1.5%. This seems to be logical because normalizing the feature vectors significantly reduces the algorithms' sensitivity to length. The modified bisecting k-means++ algorithm performed the best by producing approximately 4% better entropy and 7% better purity than the base k-means algorithm. The bisecting k-means++ algorithm performed the best most likely because it combines the strengths of both the k-means++ algorithm and the bisecting

[6] http://www.summarization.com/mead/

k-means algorithm. As suggested by [13], [14], the bisecting k-means algorithm did work better than the standard k-means algorithm. Though the k-means++ algorithm performed better than the standard k-means algorithm, the k-means++ algorithm did not perform as well as expected considering the tests performed in [15]. This may be due to the feature vector of short microblog posts.

Though these relative values show that the k-means algorithm can be improved, the absolute entropy and purity values as seen in Table I seem to suggest that the variations on the k-means algorithm will only produce small changes in the results. Hopefully, the different implementations of the criterion optimization algorithm will provide a significant increase in performance. In addition, the computation for the feature vectors may need to be reexamined because the clustering algorithms depend on good feature extraction.

B. Summarization Results and Analysis

1) Manual Summaries: Though the manual to manual ROUGE-1 F-measure scores seem to low (F-measure = 0.3291), this can be explained by the several factors. First, the instructions for summarizing did not give any guidelines to how each person clustered except for whatever themes or topics the volunteers thought could be good clusters. Therefore, the clusters for a topic may have been significantly different from one person to another depending on how they wanted to differentiate the posts. They were not limited to simply extractive clustering either since they were allowed to abstract concepts from the posts. In addition, for some topics, there was only thematic overlap rather than specific word overlap. For example, the topic "#MM" was a topic that stood for "Music Mondays" and the tweets would simply have names of songs or names of artists. Obviously, the names of songs or artists do not tend to overlap naturally. These results also seem to agree with the low F-measure scores that computed for one sentence summaries in Sharifi's work [10].

Because choosing the specific number of clusters for the volunteers (k = 4) could have introduced bias, data was collected on whether on not the volunteer thought 4 clusters was the right size for each particular post. This was a way to gather information about how good 4 clusters seemed and for future extensions of this research. Out of the 50 manual summaries—2 summaries per topic with 25 topics—the volunteers answered that there should have been less 13 times, the same 28 times, and more 9 times. This data is summarized in Figure 2.

It seems that the number of clusters (k = 4) was about the mean but was not always the best choice for all topics. Therefore, this research could be extended to discover how the number of k could be decided more intelligently.

2) Baseline Summarizers:

a) Random Summarizer - The seemingly high F-measure of the random summarizer may possibly be explained by a few characteristics of microblog posts. First, many microblog posts about a subject use the similar words in the post so unigram overlap within all posts seems to already be fairly high. Second, the most representative post about a topic is often quoted verbatim so the random summarizer has a decent chance of selecting one of these retweeted posts. Third, the unigram overlap even among manually generated summaries was low so it is not surprising that the random summarizer agreed some of the time.

b) MEAD - Interestingly, the MEAD default summarizer did slightly worse than the random summarizer. This seems to suggest that traditional ways of summarizing do not work very well with microblog posts. The unstructured and informal nature of the posts do not correlate with the expectations of the MEAD summarizer.

c) LexRank - Though the LexRank summarizer improved the random summarizer's F$_1$-measure by about 5%, it does not seem to be significantly better than the naiv̈e random summarizer. Again, this seems to suggest that summarization of microblog posts is

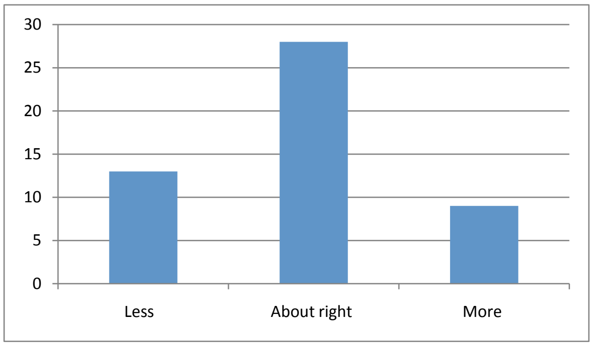


Fig. 2. Volunteer's answers to question about whether the specified number of clusters (k = 4) was right for the topic.

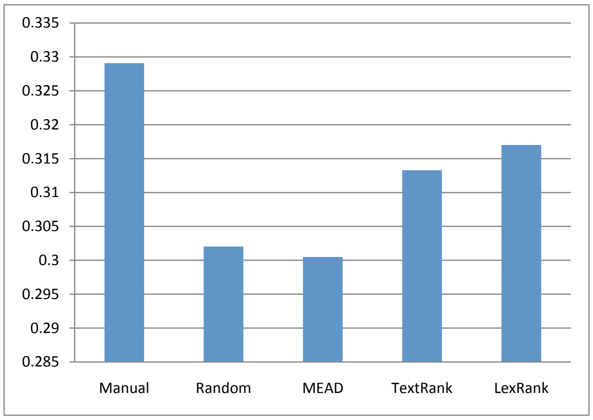


Fig. 3. F-measures for the baseline summarizers compared to the manual to manual F-measure.

significantly different than normal document summarization.

d) TextRank - Again, TextRank seemed to perform about 3.7% better than the random summarizer but not a significant improvement.

3) Cluster Summarizer: The cluster summarizer produced good results with an F-measure that is 8% better than the random summarizer. And though varying the number of computed clusters might have reduced noise, it seems from the results when $k > 4$ that the performance decreases as is shown in Figure 4. Therefore, the best Cluster summarizer is the original implementation that clustered the posts into 4 clusters and summarized each cluster into 4 representative posts.

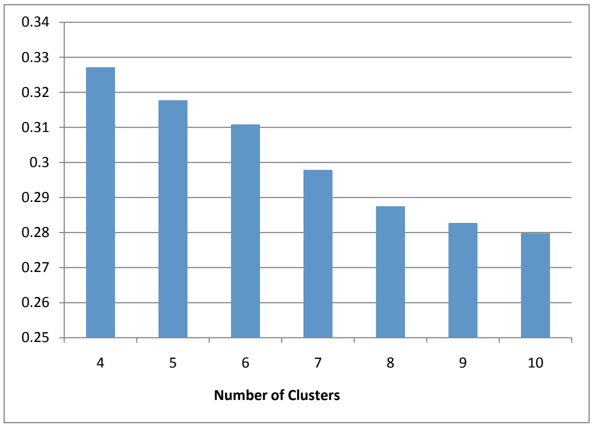


Fig. 4. F-measures of the Cluster Summarizer over the number of clusters.

4) Hybrid TF-IDF Summarizer with Similarity Threshold: The Hybrid TF-IDF Summarizer's performance was better than expected with a best F-measure of 0.3537 that was 17% better than the random summarizer when $t = 0.77$. Its best threshold measure of $t = 0.77$ seems to be reasonable because it allows for some overlap but does not allow sentences to be nearly identical. One reason for this summarizer doing so well is that it puts all the noise posts near the bottom since they do not seem to be related to other posts. In addition, the specific weighting of sentences is probably better suited for microblog posts than most traditional weightings of sentences such as normal TF-IDF.

5) Summary of Results: A summary of the best performing summarizers can be seen in Figure 6. The average precision, recall and F-measure are scaled by the F-measure of the random summarizer (0.3020) to give a relative sense of each summarizer. The values of precision, recall and F-measure with a standard deviation σ can be seen in Table II.

It can be seen from Figure 6 that the Hybrid TF-IDF and the Cluster summarizers performed better than any of the other summarizers including TextRank and LexRank. In addition, the Hybrid TF-IDF significantly improves over the Cluster summarizer by reaching about 18% better than random.

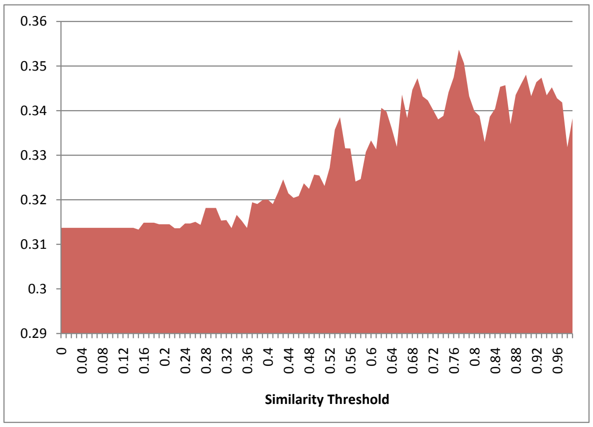


Fig. 5. F-measures of Hybrid TF-IDF Summarization algorithm over different thresholds.

TABLE II
ROUGE-1 AVERAGES

	Precision	Recall	F-measure ± σ
Manual	0.3383	0.3383	0.3291 ± 0.1089
Random	0.2885	0.3322	0.3020 ± 0.0930
Mead	0.2429	0.4109	0.3005 ± 0.0913
TextRank	0.2644	0.4075	0.3133 ± 0.0955
LexRank	0.3676	0.2943	0.3170 ± 0.0871
Cluster	0.3092	0.3606	0.3271 ± 0.1060
Hybrid TF-IDF	0.3505	0.3723	0.3537 ± 0.1172

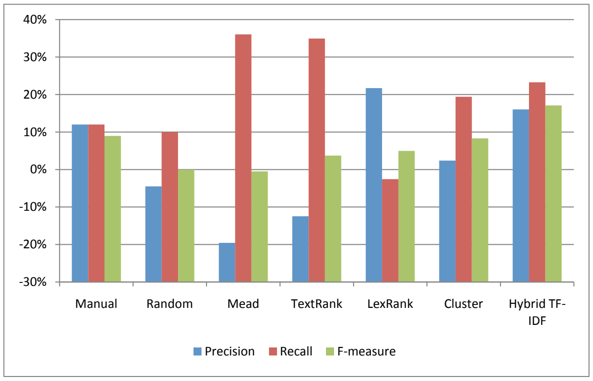


Fig. 6. Scaled F-measures of the summarizers.

Because the topic phrase will be included in every post, it seems that a unigram match of the topic phrase is actually trivial and could be hiding non-trivial unigram overlap. Therefore, the ROUGE scores were recomputed so that the computation ignored keywords. The results are shown in Table III and summarized in Figure 7. Again, the results shown in Figure 7 are scaled by the F-measure of the random summarizer (0.2071).

The drop of almost all the averages by about 0.1 when keywords were ignored seems to be about right since the average length of a post is 11 words and each post will have one of the keywords at least once ($1/11 \approx 0.1$). However, the relative results of the summarizers changed. Using this slightly modified ROUGE metric, LexRank performs less than the random summarizer and the TextRank summarizer performs just slighltly better than the Cluster summarizer. The Hybrid TF-IDF summarizer continues to significantly outperform all other summarizers with an F-measure of 0.2524 which is 22% better than the random summarizer.

TABLE III
ROUGE-1 AVERAGES (KEYWORDS IGNORED)

	Precision	Recall	F-measure ± σ
Manual	0.2320	0.2320	0.2252 ± 0.0959
Random	0.1967	0.2283	0.2071 ± 0.0817
LexRank	0.2333	0.1894	0.2027 ± 0.0760
Mead	0.1771	0.3050	0.2204 ± 0.0738
Cluster	0.2180	0.2554	0.2310 ± 0.0891
TextRank	0.1954	0.3053	0.2328 ± 0.0799
Hybrid TF-IDF	0.2499	0.2666	0.2524 ± 0.0906

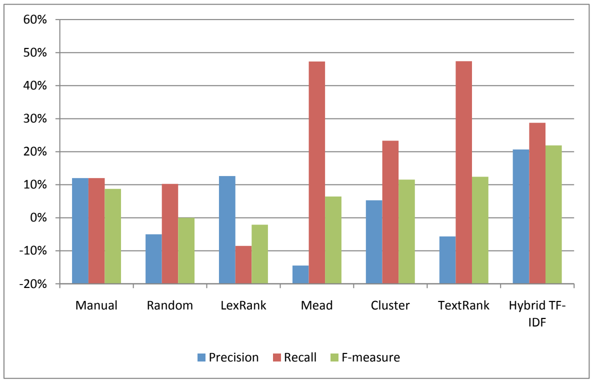


Fig. 7. Scaled modified F-measures (keywords ignored) of the summarizers.

Since the number of unigrams in the summary could affect the ROUGE scores, the average number of characters for each summarizer is shown in Figure 8. The high values of the TextRank and MEAD summarizer that are approximately 50% higher than the manual summaries, would explain why the recall values of the TextRank and MEAD summarizer are particuarly high. In addition, the results help explain why the recall of every summarizer except the LexRank summarizer are higher than their corresponding precision measures. An extension of this work may be to attempt to penalize longer posts especially for the MEAD and TextRank summarizers to see if it improves their F-measures.

Examples of the top 3 summarizers (TextRank, Cluster and Hybrid TF-IDF) appear in Tables IV-VI. The three topics were chosen based on the F-measure scores of the Hybrid TF-IDF summarizer for it's best, worst and average topic.

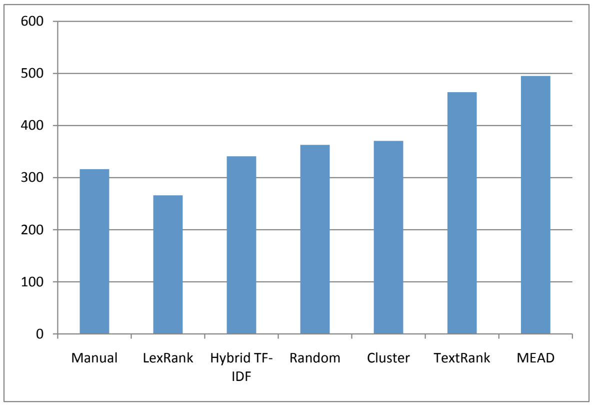


Fig. 8. Average number of characters per summarizer.

VII. CONCLUSION

The final goal of this research is to produce multiple post summaries of particular topics discussed in microblog posts in order to simplify and understand the information that microblogs provide. This research project specifically extends the work of [10], which only considered producing one sentence summaries. It seems from the clustering results that clustering microblog posts is not as simple or clean as clustering normal structured documents, and therefore, some new ways of clustering or computing feature vectors could be explored in future work.

The Hybrid TF-IDF summarizer with a similarity threshold of 0.77 produces significantly better results than the random summarizer and seems to be competitive with manually generated summaries. In addition, it performs better than some of the more traditional multi-document summarization systems MEAD, LexRank and TextRank. This points to the fact that microblog posts cannot be treated as traditional documents.

This project could be further extended in several ways. First, if a list of the most significant current topics could be computed as is being researched by [21], a summary of all the most significant topics could be generated in real time. It may also be possible to produce a topic browsing and summarization tool that will help people have a more comprehensive idea about real time microblog information.

Second, the coherency of the multiple post summary could be researched in depth. Sophisticated methods for ordering standard documents have been explored by [22], [23], and these advanced methods could possibly be applied to microblog summary cohesion. Other coherence issues such as pronoun resolution and fragmented arguments are issues that all summarization techniques need to consider [5]. If these issues could also be solved, clean, cohesive and comprehensive summaries of specified microblog topics could be practical and beneficial for many people.

TABLE IV
Gossip Girl (Best Topic for Hybrid TF-IDF)

Manual 1	about to watch gossip girl! great episode of Gossip Girl tonight! Not happy with that episode at all. #Gossip Girl gossip girl was way dramatic tonight blair and chuck can not break up
Manual 2	Yeah, it’s time for Gossip Girl! great episode of Gossip Girl tonight! Gah, this week’s Gossip Girl broke my heart in about 16 different ways. Chuck and Blair better make up soon! Just missed #gossip girl ....oh why? Oh, wait, i know.....too mauch damn homewrok!!
TextRank	i wanted to watch gossip girl with the girls but tummy ache :(this hasn’t happened in months. ugh. house finally came on and it was gooood. So... I’m sitting in Indie’s room with a bunch of Lesbians trying to explain the magnificence that is “Gossip Girl“ Aww, Chuck & Blair are mad at each other. Usually Gossip Girl ends on a good note for Chuck and Blair. This episode didn’t. I’m sad now :’(I will seriously stop watching “Gossip Girl“ if Chuck and Blair break up. And WTF? Get Hilary Duff off of the show!
Cluster	has enjoyed throwing some MST3K style riffs at Gossip Girl tonight. Drew some inspiration from Bob Evil & Nick from Time Chasers. :D #fb gossip girl was way dramatic tonight blair and chuck can not break up Hmm. Gossip Girl voice overs starting to sound an awful lot like Meredith Grey’s voice overs. This is not a positive development. wait people still watch gossip girl? lmao
Hybrid TF-IDF	Not happy with that episode at all. #Gossip Girl wait people still watch gossip girl? lmao great episode of Gossip Girl tonight! gossip girl was way dramatic tonight blair and chuck can not break up

TABLE V
#MM (Worst Topic for Hybrid TF-IDF)

Manual 1	its still monday?! welp #MM Tank “Slowly“ -hmmmmm #MM “We Used To Vacation“ Cold War Kids ¡– Pure Talent! Another one of my favs... #ralphlauren I got so many horses bitches call me polo... Guess the artist who said those lyrics #mm #musicmonday @Firefly2020 Thank you for #MM hugs - same back to you!
Manual 2	#MM The Feelies - The Good Earth #MM Keep It Flowin- Isley Brother...I could listen to this ALL day! Get up on it! ’Take your time when you likin a guy Cause if he sense that your feelings too intense, it’s pimp or die...“ #MM Jay-Z Soon You’ll Understand iphone app which might mitigate this winter by starting the car from anywhere #iPhone #automobile #MM #app #apps http://bit.ly/6yZPE
TextRank	#MM Kanye West “See You In My Nightmare“... I Got The Right To Put Up A Fight!!! That is a sad discovery!! RT @joeyt2k just found out The Darkness broke up in 2006, is too devastated to speak. #MM #musicmonday #MusicMonday I always smoke dro, so it must be the answer, best beat in the game? my votes for #BeatCancer www.myspace.com/dezine420 #mm #MM #MusicMonday This is so very gay, but Miley Cyrus “Party in the U.S.A.“ is actually starting to grow on me...
Cluster	RT @aFOOLwperspctve: #MM Bobby Womack “If u think u lonely now“ wheeeeeeeeew my shit (Mine too!!) #MM Blade Icewood - Oh Boy, it’s a detroit thing yall wouldnt understand lol #MM “Deosnt Mean Anything“ by Alicia Keys...i love dis chick #MM Amy Winehouse Black to Black album...5stars
Hybrid TF-IDF	#MM Amy Winehouse Black to Black album...5stars @young_gab...stole my #MM song #MM “Deosnt Mean Anything“ by Alicia Keys...i love dis chick RT @RoseGold88: #MM Rell feat jay Z-Love for free***thats my favorite song sis!!!

TABLE VI
A-Rod (Average Topic for Hybrid TF-IDF)

Manual 1	A-Rod and ARod are trending now...hahaha. Yankees fans: No matter how this postseason turns out, please shut up about A-Rod being a postseason choker. Yours in Christ, SDC Nice CC!!.. Posada’s off his game tonight but A-Rod’s on point! LET’S GO YANKEES! A-Rod is superman
Manual 2	A-Rod is just on fire this postseason. A-Rod homers in third straight game http://bit.ly/168LMB I HATE A - ROD TOO MUCH!!! WHO’S WITH ME? not only is ARod a trending topic but so is A-Rod lol
TextRank	Come on, Angels. Do work. Do something. Gosh, I haaaaaaate the Yankees. And A-rod is NOT worth that contract. Girardi is now going to pinch hit for A-Rod 3-2 here because that’s what his book says is the right move (via @NoYoureATowel) Wow both A-Rod and ARod is on Trending topics. Stupidddd,,, Wow both A-Rod and ARod is on Trending topics. Stupidddd,,,
Cluster	I have no idea who none of these players are besides A Rod and Derek Jeter -_- A-Rod homers in third straight game: A-Rod homers in third straight game http://bit.ly/168LMB Gotta love that both arod and A-Rod are trending: Gotta love that both arod and A-Rod are trending LOL no one is in this game. Posada leaves home plate after Jeter’s double play thinking it was 3 outs. kudos 2 A-rod who ran to cover home.
Hybrid TF-IDF	RT @johnnnyAa love this A-Rod guy, dude can really play baseball watching a-rod tie howard and gehrig’s postseason rbi streak record. howard also tied gehrig’s 70+ year old record just this year. Gotta love that both arod and A-Rod are trending: Gotta love that both arod and A-Rod are trending A-Rod homers in third straight game: A-Rod homers in third straight game http://bit.ly/168LMB

REFERENCES

[1] H. Luhn, "The automatic creation of literature abstracts," *IBM Journal of research and development*, vol. 2, no. 2, pp. 159–165, 1958.

[2] H. Edmundson, "New methods in automatic extracting," *Journal of the ACM (JACM)*, vol. 16, no. 2, pp. 264–285, 1969.

[3] K. Mahesh, "Hypertext summary extraction for fast document browsing," in *Proceedings of the AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, 1997, pp. 95–103.

[4] K. Knight and D. Marcu, "Summarization beyond sentence extraction: a probabilistic approach to sentence compression," *Artif. Intell.*, vol. 139, no. 1, pp. 91–107, July 2002. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(02)00222-9

[5] U. Hahn and I. Mani, "The challenges of automatic summarization," *Computer*, pp. 29–36, 2000.

[6] N. Madnani, D. Zajic, B. Dorr, N. Ayan, and J. Lin, "Multiple alternative sentence compressions for automatic text summarization," in *Proceedings of the 2007 Document Understanding Conference (DUC-2007) at NLT/NAACL*. Citeseer, 2007, p. 26.

[7] E. Gonzàlez and M. Fuentes, "A New Lexical Chain Algorithm Used for Automatic Summarization," in *Proceeding of the 2009 conference on Artificial Intelligence Research and Development: Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence*. IOS Press, 2009, pp. 329–338.

[8] W. T. Visser and M. B. Wieling, "Sentence-based summarization of scientific documents the design and implementation of an online available automatic summarizer," 2008.

[9] J. Kupiec, J. Pedersen, and F. Chen, "A trainable document summarizer," in *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1995, pp. 68–73.

[10] B. Sharifi, M.-A. Hutton, and J. K. Kalita, "Automatic microblog classification and summarization," 2010.

[11] G. Salton, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988. [Online]. Available: http://dx.doi.org/10.1016/0306-4573(88)90021-0

[12] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (Prentice Hall Series in Artificial Intelligence)*, 1st ed. Prentice Hall, February 2000. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0130950696

[13] Y. Zhao and G. Karypis, "Criterion functions for document clustering: Experiments and analysis," 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.3151

[14] M. H. Dunham, *Data Mining: Introductory and Advanced Topics*. Prentice Hall, August 2002. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0130888923

[15] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. [Online]. Available: http://portal.acm.org/citation.cfm?id=1283494

[16] D. Radev, T. Allison, S. Blair-Goldensohn, J. Blitzer, A. Çelebi, S. Dimitrov, E. Drabek, A. Hakim, W. Lam, D. Liu, J. Otterbacher, H. Qi, H. Saggion, S. Teufel, M. Topper, A. Winkel, and Z. Zhang, "MEAD - a platform for multidocument multilingual text summarization," in *LREC 2004*, Lisbon, Portugal, May 2004.

[17] D. Radev and G. Erkan, "Lexrank: graph-based centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–480, 2004.

[18] R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," in *Proceedings of EMNLP*. Barcelona: ACL, 2004, pp. 404–411.

[19] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine* 1," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[20] C. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004)*, 2004, pp. 25–26.

[21] J. Benhardus, "Streaming trend detection in twitter," 2010.

[22] R. Barzilay, N. Elhadad, and K. McKeown, "Sentence ordering in multidocument summarization," in *Proceedings of the first international conference on Human language technology research*. Association for Computational Linguistics, 2001, p. 7.

[23] M. Lapata, "Probabilistic text structuring: Experiments with sentence ordering," in *Proceedings of the annual meeting of the Association for Computational Linguistics*, 2003, pp. 545–552.

Syntactic Normalization of Twitter Messages

Max Kaufmann

***Abstract*—The use of computer mediated communication such as emailing, microblogs, Short Messaging System (SMS), and chat rooms has created corpora which contain incredibly noisy text. Tweets, messages sent by users on Twitter.com, are an especially noisy form of communication. Twitter.com contains billions of these tweets, but in their current state they contain so much noise that it is difficult to extract useful information. Tweets often contain highly irregular syntax and nonstandard use of English. This paper describes a novel system which normalizes these Twitter posts, converting them into a more standard form of English, so that standard machine translation (MT) and natural language processing (NLP) techniques can be more easily applied to them. In order to normalize Twitter tweets, we take a two step approach. We first preprocess tweets to remove as much noise as possible and then feed them into a machine translation model to convert them into standard English. Together, these two steps allow us to achieve improvement in BLEU scores comporable to the improvements achieved by SMS normalization**

I. Introduction

TWITTER is a relatively new hybrid micro blogging/social networking website where users can post and read messages from a variety of electronic medium, such as Twitter's own website, text messages, or their computer desktop. Twitter is a popular medium for broadcasting news, staying in touch with friends, and sharing opinions. Since its initial founding in 2006, it has obtained over 100 million users [15]. Tweets, a term used to describe messages sent on Twitter, contain only 140 characters, 20 characters less than the 160 allowed by text messages. Twitter users are not even guaranteed to be able to use all of these for content. Twitter posts frequently included URLs, as well as markup syntax, which further decreases the amount of characters available for content. Because of these limits, users have created a novel syntax, very similar to SMS lingo, to communicate their messages with as much brevity as possible. While this brevity allows tweets to contain more information, it makes them harder to mine for information, due to its lack of standardization. Table 1 shows some examples of tweets.

TABLE I
Sample Tweets

Never say never.....dont let me goo dont let mee gooo dont let me gooooo....
@user13431 when r u commin to Montreal
#bestfeeling is feeling like u mean the world to someone
My work buddy 'go smoke' like 3 times already
mai8mai RT @user1341 : Support Breast Cancer Awareness. Add A #twibbon To Your Avatar Now!!
I'm so #overyou Didn't even know it was possible!!!

There are several issues that makes the normalization of tweets a difficult task. Tweets are written extremely colloquially, containing an unusually high amount of repetition, novel words, and interjections. A word may be written using a phonetic spelling (*nite* instead of *night*), or combined with other frequently used words into an acronym (*omg* instead of *oh my god*). Twitter users also have little regard for the proper use of capitalization and punctuation. Capitalization in a tweet may signal a proper noun or a sentence boundary, but it may also be used for something as arbitrary as emphasizing a certain segment. Punctuation may signal sentence boundaries, but it might also be used to create an emoticon. There are some deviations that are standard and systematic, but new variations can be created at any time, making the process of modeling the language extremely difficult. Additionally, Twitter users frequently use symbols to encode meta-content, such as who the tweet was directed to, or the topics to which it pertains. This meta-content sometimes is integrated into the syntax of the tweet, but there is no guarantee that it will be. In order to normalize these tweets, they will first be preprocessed to remove as much of the noise as possible, then fed into a machine translation model to convert them into standard English.

II. Motivation

Due to Twitter's popularity, it has produced a massive amount of data. This data offers new and exciting opportunities, and there is much useful information that can be learned from meaningful analysis of this data. But the quality of the data is so poor that standard NLP tools are unable to process it. Tools such as Named Entity Recognizers have been shown to perform extremely poorly on tweets, most likely due to the high amount of noise present in tweets [5]. It has been shown that normalizing text messages allows standard MT techniques to work on them with little or no adaptation [3], and this paper posits that the same is true for tweets. If tweets can be converted to standard English, then the same should hold true for them. Another area in which data from Twitter can be used in is trend analysis. [4] claims that the unstructured nature of news articles, and the difficulty of NLP makes the problem of finding trends and topics in news articles a rather complex problem. These issues are magnified in tweets, which have all of the issues that normal news articles do, in addition to non standard orthography and extreme noisiness. Normalizing tweets would make work in this area, as well as any other area that involved analyzing tweets, much easier.

This is not to say that nobody has had success in mining data from tweets. Many studies have been able to draw conclusions from analyzing data on Twitter. Studies such as [2] have investigated how individuals use Twitter to communicate vital information in states of emergency. Papers such as [17] have shown that the informal communication that microblogs foster improves collaboration in the workplace. However, these studies concern the social effects of Twitter. Studies such as Puniyani et al. which attempt to preform an analysis focused on the content of the tweets admit that “Twitter contains highly non-standard orthography that poses challenges for early-stage text processing” [14].

III. Previous work

While normalization of Twitter posts has never been attempted before, work has been done on noisy text normalization in the NLP field. However, tweets have several properties which makes normalizing them a substantially different problem than normalizing other forms of noisy text, such as emails or forum posts. First, they are very brief, containing only 140 characters. This means that it is much more difficult to use context as part of the disambiguation process. Tweets also have several novel syntactic elements which are especially challenging to disambiguate. Despite these differences, the process of tweet normalization is actually fairly close to the process of SMS normalization.

One area in which SMS normalization has been approached is to compare it to speech recognition. Text messages contain a significant number of tokens that are more indicative of its pronunciation, rather than its normal orthography.(e.g., *rite* instead of *right*)[9]. Speech recognition techniques are designed to decode phonetic representations into written words.[9] used techniques from automated speech recognition in order to normalize SMS. [9] claims that the dynamic nature of SMS is very difficult to capture with only a rule based MT system. They encoded the tokens in SMS messages into phonetic forms, and attempted to find the correct word with the most phonetic similarity.

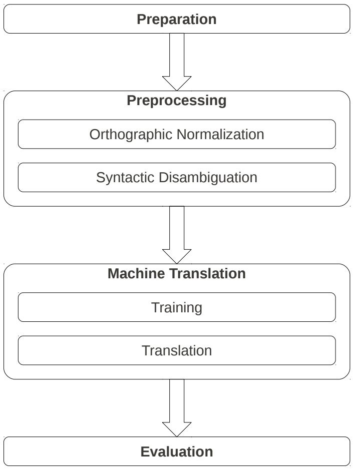
Another way to approach the problem is to look at tweets as though they are a different language, and attempt to use machine translation techniques to normalize them. This approach is fairly popular. Kobus et al [9] used this approach in addition to phonetic decoding. Others such as [8] and [3] have used supervised learning machine translation models to attempt to capture the most common SMS phrases and their English equivalents in a phrase table.

IV. Normalization

Previous work such as [9] has suggested that combining multiple approaches in noisy text normalization creates the best results. Additionally, normalizing input text before inserting it into a MT system has been shown to improve the output quality. [13] modified their translations to harmonize word order, in order to improve the accuracy of their MT system. Preprocessing the tweets decreases the amount of noise present when they are being analyzed by the MT system, thus increasing the quality of the results. There are several issues, discussed in the following sections, which are easily dealt with by preprocessing, but would cause a great deal of confusion to a MT system.

The normalization model consists primarily of two parts, a normalization module and a statistical machine translation module. Below is a diagram.

Figure 1: Tweet Normalization Process



A. Preparation

Before beginning the experiment, it was necessary to hand annotate tweets, so that there was a gold standard to evaluate the quality of the translation against . Approximately 1 million tweets were extracted from the the Edinburgh Twitter Corpus, a corpus containing 97 million Twitter posts[1]. These tweets were then filtered to remove tweets that were not in English. A tweet is not considered to be in English if it has under 40% English words. From these tweets, 1150 were randomly selected. These were hand translated by 10 annotators. The goal of this project is to remove as much noise as possible from tweets, so the annotators were instructed to remove any elements that were not absolutely necessary to form a grammatical English sentence. This included deleting elements such as smilies and extraneous punctuation, inserting subject pronouns, replacing acronyms, and correcting verb tenses. For example, the tweet *@user213 how are you?? I’m good :*” would have been translated as *How are you? I’m good*. This is a significantly different approach than is normally taken with SMS normalization. In SMS normalization corpora such as the ones created by [8], [6], many extraneous elements were kept in. The decision to remove them was motivated by the goal of this project. In order for tools such as named entity recognizers and semantic role labelers to work, they need their input to be as close to standard English as possible, and so it makes sense to remove these elements in tweets. Extra information would only serve to confuse these tools. In the normalization model, most of the extraneous elements are removed during preprocessing, so if a study were interested in using this extraneous information, it would be a fairly trivial task to leave these elements in.

B. Orthographic Normalization

Although, Twitter messages are similar syntactically to SMS messages, they differ significantly in their type and quantity of orthographic errors. SMSs are created almost exclusively on cell phone, while approximately 90% of tweets come from the Web, IM, or custom applications, according to [11], all which use an automated spellchecker to suggest spelling corrections to the user. Because text messages are not spell checked prior to their submission, they are most likely going to have many more unintentional errors. The orthographic normalization model assumes that the main source of error in tweets will be from intentional spelling errors, since the spellchecker will have taken care of the majority of accidental errors. However, identifying spelling errors in Twitter messages is a difficult task. Twitter is frequently used as a medium to broadcast news [11] and therefore contain a large number of proper nouns that are not likely to be contained in a dictionary. Figuring out whether a word that is not in the dictionary is a misspelled or simply a novel word is not a trivial task. Because of this, the orthographic model is fairly conservative in spelling correction, to avoid misidentifying a novel word as a misspelled word. The approach taken simply identifies and corrects the most common intentional orthography errors.

One of the most common types of orthographical errors in Twitter posts is shortening words. Frequently used phrases are shortened into acronyms, and frequently used words are shortened by using phonetic spellings, or having characters removed. To help disambiguate these terms, a table of common SMS acronyms and short forms was created. This table was based on the work by [6]. [6]created a table containing common SMS errors and their English equivalents. This list was parsed to obtain a list of SMS acronyms that could be directly mapped to English words of phrases. The original table contained ambiguous mappings, such as translating *wt* as *what*, even though it is sometimes translated as *with*. If a statistical MT system were not part of the normalization approach, it might have been a good idea to leave the ambiguous mappings in, and just replace errors in the tweets with their most common correction. However, the statistical MT model is capable of disambiguating based on context, and can resolve ambiguous mappings. Therefore, all ambiguous sms terms which had multiple English mappings were stripped from the list by hand, leaving only items that could be unambiguously mapped to an English equivalent, such as *u, 2moro*, and *wut.*

One of the easiest spelling errors to make are off-by-one transpositions. For each misspelled word, all possible combinations that involve swapping two adjacent letters are tried, and if a correct match is found in the dictionary, the correct spelling is substituted.

Twitter messages frequently use repetition to convey emphasis. Written text lacks the tonality and variations which are used to convey emotions in spoken language, and so Twitter users are forced to creatively find ways to express emotion in their limited 140 characters. Similar to the way that people drag out words in spoken language to emphasize them, Twitter users frequently repeat characters in order to create emphasis. For example, a Twitter user wrote *OMG! I'm so guilty!!! Sprained biibii's leg! ARGHHHHHH!!!!!!* The repeated exclamation marks and extra letters on the token *argh* serve to emphasize the author's emotions. To correct for this, misspelled words that contain repeated sequential letters have these letters removed. If removing these letters creates a correct word, than the misspelled word is replaced in the text. Similarly, repeated punctuation is shortened to one punctuation mark, since the additional punctuation marks are not syntactically necessary.

C. Syntactic Disambiguation

1) @: There are several elements in tweets that only sometimes have syntactic value. One of the elements is @username. Typing "@username" in a tweet is a processes commonly referred to as replying. However, the term "replying" is somewhat of a misnomer, as a user does not need to receive a message in order to "reply" with this syntax. The most general definition of this symbol is that it means the author of the post is telling a certain user that he thinks they would be interested in the content of the tweet. The most common use is to preface a tweet with @username (e.g., @Sammy wanna go to the park?). However, @username can appear at any point in the tweet. It can appear in the middle, if the author wants to address different sections of the tweet to multiple people (e.g., @sammy I'll be over tommorow @sally I'll fix it later). In these situations, @username has no syntactical value. However, that is not always the case. The @username is frequently incorporated into the sentence. For example, a user may write, *@sammy is my best friend*! or *Im at the park with @sammy and we're having a great time*. In these situations, the @username performs a syntactic role in the sentence, and its removal would be grammatically improper.

Another feature of the @username syntax is that it can be used to broadcast information to all of a users followers. Twitter users frequently send tweets they find interesting to all of their followers with the syntax "RT @username:", where username is the username of the original author, of the followed by the original message. While this appears similar to @username, it is syntactically different. RT @username almost always has no syntactic value, and can be removed while maintaining proper syntax.

Analysis of these tweets has revealed that there are certain linguistic properties that can be analyzed to remove the majority of @username when it is appropriate. By tagging the text that is being translated with part of speech (PoS) tags, it becomes apparent that when the @username needs to be kept in the sentence for syntactical reasons it is preceded or followed by certain parts of speech. If the @username is at the beginning of the tweet, then only the subsequent terms can be used in this analysis. If the @username is followed by a word that is either a coordinating conjunction, subordinating conjunction, preposition,or a verb it is almost always necessary to keep the @username in the tweet. If it is not the first word, then the part of speech of words on both sides can be used to help disambiguate @username. In these situations, the preprocessor checks for the above conditions, but it also checks to see if the part of speech of the preceding word belongs to the previous list.

2) #: Another element which may or may not have syntactic value is the #. The most common syntax of # (read as hash or hash symbol) is #topic. The word following the # is generally the topic to which the tweet pertains. If a user was tweeting about the government stimulus bill, they might insert *#stimulus* into their tweet. This process is called tagging, and a #topic is commonly known as a tag. Twitter uses these tags to classify posts by common topic. Like @, the # may or may not have syntactic value. It is most commonly inserted at the end of a tweet without any syntactic value (e.g., I just got the new Droid phone #droid), but if the topic of the tweet is contained within the tweet, users frequently append a hash to the topic, in order to stay within the 140 character limit and avoid repetition (e.g., I just got the new #droid phone). Additionally, in tweets that are about the user's mood instead of a certain topic, it is common to use the # for emphasis (e.g., At work thinking abt how I have to leave tonight and come Right back in the am. #Argh And I have to train somebody 2day).

Unfortunately, PoS tags alone cannot be used to decide whether a hashtag has syntactic value in the sentence. @username is always a noun, whereas a hashtag can be any type of word. Observation of the annotated tweets shows that humans almost always thought that terms with a hashtag located in the middle of a sentence were important to the syntax of the sentence, and should not be deleted. However, hashtags at the beginning and end of tweets are much more difficult to disambiguate. This is an unfortunate problem, since the beginning and end of tweets are where hashtags are most commonly found. To disambiguate these hashtags the following heuristics were used: If there are two or more sequential hashtags, it is likely that they are topics, and have no syntactic value, and so they can be removed. Additionally, if a hashtag is preceded by a terminal punctuation mark, we can assume that they are standalone topics, and play no role in the syntax of the tweet, and can also be removed. If the hashtag is preceded by a conjunction, preposition, or transitive verb, then we can assume that the hashtag is syntactically linked to the previous term and needs to remain in order to preserve the syntax of the tweet. If none of these conditions were met, then the hashtag was removed.

D. Statistical Machine Translation

After the preprocessing is done, the tweets are ready to be fed into the statistical machine translation system. The tool that was used to build this system is Moses. Moses is a statistical machine translation package which can produce high quality translations from one language into another[10]. At its core, translation simply consists of finding phrases in one language that correspond to phrases in another language. While the tasks of tweet normalization is not translation, it does consist of converting one set of phrases into another set, which makes Moses an extremely valuable tool.

1) Training: According to the Moses website [1], there are 9 steps involved in creating a Moses model

1) Prepare data
2) Run GIZA++
3) Align words
4) Get lexical translation table
5) Extract phrases
6) Score phrases
7) Build lexicalized reordering model
8) Build generation models
9) Create configuration file

GIZA++ is a tool which attempts to align the words from one corpus which their equivilant, or equivilants in another. When translating from one language to another, this is a difficult task, since one word in one language may correspond to several words in another. However, when translating from tweets to normal English, this is a fairly trivial task, since most of the words have a one to one mapping. Step 3 simply uses heuristics to increase the accuracy of the word alignmetns suggested by Giza. The result of all this is step 4, a lexical translation table. This table simply gives the probability for w(e|t), where e is an English word, and t is a word in Twitter English. Based on this lexical translation table, and the alignments created by GIZA++, Step 6 can created a phrase translation table, which is similar to the lexical translation table, except that it contains the probabilities of phrases in a tweet being translated as a particular English phrase. Steps 7 and 8 refer to steps which are relevant to the change in word order that comes from translation, and reverse translations. These steps are not relevant to the processes being discussed in this paper and so will not be discussed.

Before Moses can be used to produce translation, it must be trained on a data set, so that it can learn the rules that govern the translation. Training Moses requires a corpus in the target language, from which an *n-gram* language model (LM) is built. In this experiment, the LM was built from the Open American National Corpus (OANC)[7], a corpus of 15 million words from a variety of contexts. Moses also requires a set of parallel corpora, one in the source and one in the target language. This posed a significant problem, since there are currently no annotated tweets which could be used as corpora. To resolve this issue, a set of parallel SMS corpora was used. These corpora were created by [8], who generously made them available for use in this experiment. The corpora consist of approximately 18,000 text messages, gathered from various sources. They were annotated by the two authors of [8], who did not use inter-annotator agreement to validate their results. While this would be an issue if this were a translation problem, normalizing text messages and Twitter posts is a much more akin to correcting grammar than translating text from one language to another, and so inter-annotator agreement is not necessary.

2) Translation: Since the task at hand is not truly translation from one language to another, several of Moses' default settings have to be tweaked in order create a high quality translation. The first is the distortion limit. Translating from one language to another often requires heavy reordering of the words. By default, Moses will allow the reordering of phrases up to 7 words long. This feature would be useful in a situation involving translating into a target language that had a word order very dissimilar to the source language. However, in this context, Twitter English lines up at almost a one-to-one ratio with normal English. Several settings were tested, and

[1] http://www.statmt.org/moses/?n=FactoredTraining.HomePage

the results indicated that the distortion limit made very little difference, similar to the findings in [8]. They used Moses for the normalization of text messages, and found that most of the phrases learned by the system only involved a one-to-one mapping. So even when the translation model was allowed to alter the word order, it chose not too.

Each element of the translation module of Moses is weighted. The weights of the translation model tell Moses how much emphasis should be placed on certain factors, such as the *n-gram* ordering derived from the language module, in the translation process. By adjusting the weight of the language model (LM), it was found that the BLEU scores could be increased by approximately .4 if the LM had a weight of .3, instead of its default weight of 1. This means that the *n-grams* generated from the OANC were not considered very important when translating. This makes sense, given that there is very little overlap between the domain of the OANC and the domain of this project.

An additional feature in Moses is the recaser. The recaser was originally designed to correctly case text if it had been translated in lowercase. In previous experiments on SMS normalization, the issue of case was ignored. However, it is very important in tweet normalization, because tweets contain so many proper nouns. In this experiment, the recaser was trained on the LM built from the OANC. When the tweets were originally translated into English, all previously seen tokens were lowercased. This was because the original capitalization of a tweet is not a reliable indicator of the true capitalization. Twitter users frequently use capitalization as emphasis, by either capitalizing the entirety of a word, or the first letter of a series of words. Unknown tokens were left in their original case, because they were most likely to be proper nouns or acronyms.

The recaser uses techniques similar to those outlined in [12]. This involves building a trigram language model, and using that to compute the probabilities of the most likely case [12]. For example, *new* is almost always lowercased, but when it is followed by *York*, it is almost always capitalized. This technique seems to be very successful in correctly casing the text, except when commonly seen words appear in a novel sequence that requires them to be capitalized. For example, the tweet *@user1941 The film I really want to see at the mo is Men Who Stare at Goats* was translated as “The film i really want to see at the moment is men who stare at goats”, because the tokens *men, who, stare,* and *goats* were almost exclusively lowercased in the OANC. However, in situations where Twitter users forgot to capitalize commonly used proper nouns, the system preformed very well. The tweet *HOME ALONE RT @user3413 : I'm craving for christmas movies!! any suggestion??* was translated as *Home alone I am craving for Christmas movies! Any suggestion?*, successfully lowercasing the capitalized text at the start, and uppercasing the proper noun “Christmas.”

V. Evaluation

The goodness of a translation is judged is using the BLEU score. The BLEU score is a tool designed for evaluating the accuracy of translations from one language to another. A BLEU score requires a gold standard, which contains the translations as done by human. This file is compared against a machine translated version, and is then assigned a score between 0 and 1. A score of 1 would indicate that the machine translated version is exactly the same as the human translated version, while 0 means that the two versions are very different. The language of a tweet is so different from the normalized result that this tool should provide an accurate indication of how well the translation worked. Below are the BLEU scores of the translation before and after normalization. NIST, an alternate MT scoring metric, scores are included in the table, so that future papers who choose to evaluate their work with NIST will have a baseline to compare their results against.

TABLE II
Evaluation of results

	BLEU scores	NIST scores
Before Normalization	0.6799	10.5693
After Normalization	0.7985	11.7095

The results indicate that the normalization process had a significant effect on BLEU scores, increasing them by 18%. Since Twitter normalization has never been undertaken before, there are no results against which these scores can be compared. The closest available data is data regarding SMS normalization. Below is a table of several papers on SMS normalization and the BLEU scores they achieved.

TABLE III
SMS Normalization scores

	Kobus et al.	Karthik and Krawczyk	Chourhury et al.
Before Normalization	n/a	.54	.57
After Normalization	approx. .8	.8611	.8070

These data shows that tweets appear to have much higher initial BLEU scores. This supports the theory that tweets do not have as many orthographical errors as SMS messages. Acronym usage and misspelling are much more common in SMS than in tweets. This is demonstrated in [8], where they were able to boost the BLEU score of a test set from .562 to .794 by simply performing a search and replace for common texting acronyms. In comparison, the preprocessing module of this normalization system, which included a very similar search and replace, only boosted the initial score of .6799 to .7324. This seems to suggest that while SMS may contain more errors to begin with, they contain a far more regular pattern of errors. The difficult part of tweet normalization is removing noisy elements which prevent the tweet from being mapped to an English syntax. So, while SMS and tweet normalization are attempting to achieve similar goals, the problems they attempt to solve are actually fairly different.

A. Sample Translations.

Below are several translations generated by the normalization system. The <URL> tag signifies that the original tweet contained a URL in that location.

Original: @user3419 nay lol y u say dat?&wat u doing 2day?

Post-normalization: No, why did you say that? What you doing today?

Original: 1001 colors: Contemporary art from Iran <URL> #Iran #culture #Art

Post-normalization: 1001 colors: contemporary art from Iran <URL>.

Original: it's soo quiet, it's like I'm goin die

Post-normalization: It is so quiet, it is like i am going to die.

Original: #worstfeeling buyin a fresh laptop..then ur screen blowz out :((

Post-normalization: worst feeling is buying a fresh laptop.. then your screen blowz out.

Original:This is superb Grape+apple splash with manggo juice, super!

Post-normalization: This is superb grape + Apple splash with mango juice, Super!

Original: @user31903 u n ur fam can n if u interested ill b n touch w u bout it

Post-normalization: You and your family can and if you interested Ill be and touch with you about it.

Original: RT: @user4191 BEAUTIFUL CREATURES has a new #website designed by @user4192!

Post-normalization: Beautiful creatures has a new website designed by @user4192!

From these tweets, it is apparent that the syntactic disambiguation module of the preprocessor is able to discern whether the syntactically ambiguous elements in the tweet are necessary or not. The model successfully removes it when it has no direct mapping to English syntax, and keeps it when it is necessary. The output of the normalization system produces much more readable results, removing extraneous noise that doesn't map to English syntax.

However, there are some issues with the normalization system. There are some orthographic errors that are not caught. The orthographic normalization system does not deal with phonetic substitutions very well, such as *blowz* instead of *blows*. Dealing with these requires a very sophisticated model, such as the one created by [6]. We feel that these errors are rare enough that the additional computational complexity required by these models is not justified in this system. However, future work attempting to improve the quality of the results could implement a system like this.

VI. Possible improvements

A. Metrics

One possible area of improvement involves finding a better metric to evaluate noisy text normalization. While previous researchers studying SMS normalization have chosen to evaluate their results with the BLEU metric, it might not be the best choice. The BLEU scoring metric was designed for evaluating translations from one language to another, not for evaluating the results of noisy text normalization. Because of this, a better BLEU score does not necessarily mean a better translation. For example, the subjectivity of the human annotators could cause substantial variation in BLEU scores. In papers such as [8] their corpora was only annotated by two people. The fact that there were 10 annotators could have lead to inconsistencies in the scoring data. For example, although annotators were instructed to expand contractions, some annotators chose to translate *Im* as *I'm*, instead of *I am*. BLEU scores are obtained by comparing the similarities between *n-grams* of the hypothesized translation and gold standards, so errors such as this could have detrimental effects on the score, despite the fact that "I'm" and "I am" are grammatically equivalent.

Even if we ignore the issue of the applicability of BLEU as an evaluator itself, there are still several problems with the BLEU metric itself. The relationship between BLEU scores and human judgment is questionable. Papers such as [16] have suggested that an increase in BLEU score may not correlate with an increase in translation quality. In fact, on a test of several machine translation systems, the correlation between human and BLEU scores was found to be as low as .38 in some cases. One example where the BLEU score performed poorly was on the tweet *@user12493 I'm following u now should I hold on tight?*. The translation generated by the normalization system was *I'm following you now, should I hold on tight*". This seems like a perfectly acceptable translation. However, the human annotator translated the tweet as *I'm following you. Now, should I hold on tight?*. BLEU scores this translation at .43. However, both translations are acceptable.

B. Corpora

Besides improving the scoring metric, there are several ways in which the translation process can be improved. The easiest improvement would be to use tweets as training data, instead of text messages. Constructing an annotated twitter corpora would be a difficult and time consuming task, but would allow the MT model to do much of the work done in the preprocessor, such as syntactic disambiguation of @username or #tag. Providing more detailed data would also improve the quality of the results. Moses has the ability to incorporate additional lexical information such as PoS tags into its translation model. Including this information would allow Moses to create more sophisticated rules governing the translation from tweets to English.

A better language model would also improve the quality of the translation results. The current corpus used to build the language model, the OANC is not especially representative of the structure of tweets, as evidenced by the fact that decreasing its weight in the translation process from 1 to .3 resulted in an better results. Perhaps a language model built from text messages, or tweets, would be better. This study attempted to build a LM from the SMS corpora provided by [8], but it did not improve the quality of the results. However, this is probably due to the fact that the OANC contains magnitudes more data than the SMS corpus. If an SMS corpus of sufficient size could be obtained, it would probably create a much more applicable language model.

C. Utilizing Additional Properties

There are additional ways in which semantic information of tweets could be used to aid in the normalization process that fall outside the scope of this paper. While there are many acronymns that are standard across Twitter, there is no official standard language. Because of this, it is difficult to draw conclusions about the nature of the language used on Twitter by looking at a large set of tweets. However, it might be possible to use Moses to create localized translation models by looking at smaller subsets of tweets. For example, tweets from users who reside in a particular nation might have their own set of slang. Tweets that are obtained from Twitter include information about the users location, as well as their country of orgin (if they have elected to include that information). This information could be leveraged to create a localized corpora which can more accurately translate slang from a certain region.

There are other factors besides semantic information that could be used to improve the quality of the translations. Many papers such as [6] have used phonetic systems to do orthogrpahic normalization, and have achieved a fair degree of success. This approach could be combined with the ones mentioned in this paper fairly easily. Additionally, heuristics could be used to combine the possible outputs of a phonetic system with the results of this system to decrease overall error. For example, a phonetic system would realize that *rite* is phonetic approximation of the word *right*. This would help disambiguate between other possible spelling suggestions, such as *write*. In turn, the semantic properties of the tweet could be used to decide if the usage of *rite* is a mispelling or not.

VII. Conclusion

In this paper, it was shown that combining statistical machine translation software with a preprocessor, it is possible to remove the majority of noise from a tweet, and increase its readability significantly. The benefits of this study are a novel system which can successfully map a tweet to a syntactically correct English sentence. It seems that the results of this study are sufficiently accurate enough to allow tweets to be mined for data. Additionally, now that the tweets conform to normal English syntax, NLP tools such as part of speech taggers, document summarizers, named entity recognizers, or semantic role labelers should achieve much better performance.

The value of the work in this paper is in its applicability to other procedures. One of the applications that should so significant performance when combined with this tool is a Twitter post summarizer. David Inoyue is currently working on a program that produces multiple sentence summaraizes about Twitter posts on one topic. The resulting summarizies are made up of the tweets in that topic. Since the summaries are made of tweets, normalizing them should make them significantly more readable. David is currently in the process of measuring the success of his summarizer, and when he is done we will include the results that normalization had on this process in this paper.

VIII. Acknowledgments

The author of this paper would like to thank Karthik Raghunathan and Stefan Krawczyk for making their SMS corpora available.

References

[1] *The Edinburgh Twitter Corpus*, Los Angeles, California, June 2010. Computational Linguistics in a World of Social Media.

[2] *The Nays Have It: Exploring Effects of Sentiment in Collaborative Knowledge Sharing*, Los Angeles, California, June 2010. Computational Linguistics in a World of Social Media.

[3] AiTi Aw, Min Zhang, Juan Xiao, and Jian Su. A phrase-based statistical model for sms text normalization. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 33–40, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[4] Daniel Billsus and Michael J. Pazzani. A personal news agent that talks, learns and explains. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 268–275, New York, NY, USA, 1999. ACM.

[5] James Martin Brian Locke. Named entity recognition: Adapting to microblogging. Master's thesis, University of Colorado, 2009.

[6] Monojit Choudhury, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar, and Anupam Basu. Investigation and modeling of the structure of texting language. *Int. J. Doc. Anal. Recognit.*, 10(3):157–174, 2007.

[7] Nancy Ide and Catherine Macleod. The american national corpus: A standardized resource for american english. In *Proceedings of Corpus Linguistics 2001*, pages 831–836, 2001.

[8] Stefan Krawczyk Karthik Raghunathan. Investigating sms text normalization using statistical machine translation. Stanford University, Stanford, CA, 2009.

[9] Catherine Kobus, François Yvon, and Géraldine Damnati. Normalizing sms: are two metaphors better than one? In *COLING '08: Proceedings of the 22nd International Conference on Computational Linguistics*, pages 441–448, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[10] Phillip Koehn and Hieu Hoang. Moses: Open source toolkit for statistical machine translation. Technical report, Annual Meeting of the Association for Computational Linguistics (ACL), demonstration session, Prauge, Czech Republic, June 2007.

[11] Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt. A few chirps about twitter. In *WOSP '08: Proceedings of the first workshop on Online social networks*, pages 19–24, New York, NY, USA, 2008. ACM.

[12] Lucian Vlad Lita, Abe Ittycheriah, Salim Roukos, and Nanda Kambhatla. truecasing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 152–159, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[13] Sonja Nieben, Hermann Ney, and Lehrstuhl Fur Informatik Vi. Morpho-syntactic analysis for reordering in statistical machine translation, 2001.

[14] Kriti Puniyani, Jacob Eisenstein, Shay Cohen, and Eric P. Xing. Social links from latent topics in microblogs. In *Conference on Social Media*, page 31, June 2010.

[15] Reuters. Twitter snags over 100 million users, eyes money-making. April 2010.

[16] Ying Zhang, Stephan Vogel, and Alex Waibel. Interpreting bleu/nist scores: How much improvement do we need to have a better system. In *In Proceedings of Proceedings of Language Resources and Evaluation (LREC-2004*, pages 2051–2054, 2004.

[17] Dejin Zhao and Mary Beth Rosson. How and why people twitter: the role that micro-blogging plays in informal communication at work. In *GROUP '09: Proceedings of the ACM 2009 international conference on Supporting group work*, pages 243–252, New York, NY, USA, 2009. ACM.

Generating a Large, Freely-Available Dataset for Face-Related Algorithms

Benjamin Mears
Amherst College

***Abstract*—Research in computer vision is data intensive. Over the last decade, numerous datasets have been published, but in many cases these datasets were carefully designed for specific tasks, resulting in artificially easy sets of data.**

In particular, datasets designed for the test of face detection and face recognition algorithms often are crafted under constrained settings with predefined poses and illumination conditions. And in many cases, individual research groups craft their own datasets for testing and training, but do not make their data publicly available, perhaps out of copyright concerns. Thus, with no standard of comparison, results of different algorithms are hard to compare since they may be trained and even tested on different data. And further, much wasted effort is expeneded on gathering face images each time a research group seeks to design new face algorithms. And while to some extent these problems have been mitigated for face recognition with the recent introduction of the Labled Faces in the Wild dataset [1], the difficulties remain unabated with face detection.

In this work, we seek to provide a large, freely available dataset for face-related algorithms, and in particular face detectors. By including images from existing datasets and significantly suplementing these with images retrieved from the internet, we provide a large set of data that can be used to test and compare both existing and new algorithms.

I. Introduction

The field of computer vision requires a vast number of images to test and compare algorithms. To be useful, these images must be publicly available and organized in such a way to allow researchers to compare results. Over the last decade, numerous datasets have been released and widely utilized by the computer vision community. The original Caltech-101 dataset consisted of images of 101 different objects. Images were collected by graduate students using Google Image Search and manually processed. On average 50 images for each object class were collected [2]. More recently, a similar dataset has been expanded to 256 objects. This dataset is more challenging than Caltech-101, with less manual processing of the images [3]. In their ImageNet, Deng et al. seek to associate 500-1000 images with each synset in WordNet for a total of 50 million images. They use Amazon Mechanical Turk, a platform in which tasks can be posted for users to complete in return for monetary payment. The dataset currently consists of 3.2 million images [4].

Among face datasets, many well-known collections have been published. See Figure 2 for example images from different datasets. The FERET dataset consists of grayscale images of subjects in different poses. The dataset collection process was relatively controlled, with all faces centered, no subjects wearing glasses, and all images taken against a plain background [5]. The CMU pose, illumination, and expression database (PIE) consisted of over 40,000 images of 68 subjects taken under various illumination conditions and with different poses and facial expressions. Faces are centered in the images and the dataset is mostly aimed at testing and training face recognition algorithms [6]. More recently, a more comprehensive dataset, MULTI-PIE, has been collected containing even more subjects under an expanded number of illumination conditions and pose angles [7].

Over the past few years, the Labeled Faces in the Wild (LFW) dataset has become the standard for testing and training face recognition algorithms [1]. This set of images was derived from the Faces in the Wild dataset which was collected from Yahoo! News. Along with the images, possible name labels were extracted from the associated image captions [8]. LFW further refined the dataset in [8] by manually labeling a subset of the images [1]. While these images were indeed collected from the internet and as such contain a variety of backgrounds, illumination conditions, and scales, faces were only included if they were detected by a Viola-Jones face detector [9] provided with the OpenCV library [10]. Thus the pose of the faces included in LFW is limited by the range of poses that can be detected by Viola-Jones. Further, while multiple cascades are provided with the OpenCV library, each able to detect different, although overlaping, subsets of faces, [1] only used the haarcascade_frontalface_default.xml classifier cascade.

Compared to the number of datasets tailored to the task of face recognition, very few datasets for face detection algorithms are publicly available. One of the most well-known is the MIT-CMU dataset. It consists of images collected by researchers at CMU as part of their work in [11] and also by Sung and Poggio at the AI/CBCL Lab at MIT. Yet, this set is limited, with approximately 500 faces in around 200 images. The Caltech 10,000 Web Faces dataset contains 10,524 faces in 7,092 images and was collected by using common names in Google Image Search [12]. While containing a large number of faces, most are relatively prominent in the images and thus easy to detect. Recently, researchers collected two challenging collections of face images as part of their research in blurring faces in street view images provided on Google Maps [13]. The first set, termed "Cities Face Set" contains 1,614 faces sampled from 29,106 images taken on 162 days. Unfortuanately, due to the same privacy issues they are trying to solve, the set could not be released publicly. Thus, they instead created a second set termed the "Campus Face Set"consisting of 15,075 faces of consenting individuals. While they admit the latter set is not as challenging as the former since in the

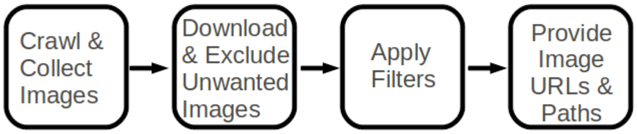

Crawl & Collect Images
Download & Exclude Unwanted Images
Apply Filters
Provide Image URLs & Paths

Fig. 1: **Image Pipeline** Images URLs are first retrieved from the internet then later downloaded and processed.

latter many of the individuals are looking direcly at the camera, the “Campus Face Set” still remains a difficult dataset with a variety of poses, scales, and in-plane rotations.

With a limited number of datasets tailored for face detection publicly available, many researchers resort to collecting their own images. In particular, while researchers may test on the publicly available datasets such as MIT-CMU in order to compare their detectors to previous approaches, a much larger dataset is required for training and so much time, effort, and financial resources must be spent on collecting a suitably-sized training set. To alleviate the burden of data collection, some resort to constructing additional synthesized face images by rotating, mirroring, or performing other manipulations on existing face images. For instance, [14] collected 10,000 images and then expanded the set to 40,000 images by mirroring, rotating, translating and scaling the existing images. In [15], they collect over 30,000 frontal, 25,000 half-profile, and 20,000 full-profile faces. Clearly such data collection requires a large amount of resources that could be better used to further research on the detectors themselves.

Recently, there has been a trend to use services such as Google Image Search or Yahoo! Images as a source for images. For instance, as part of their FaceTracer project, Kumar et al. assembled a database of over 3 million faces using online sources such as Google Images and Flickr [16]. Yet, many of these services place restrictions on the number of images retrieved. For instance, Google Images restricts its searches to the first 1000 images [17]. Various approaches have been used to get around these limitations. For instance, in addition to the images directly returned by the image search engines, [17] also downloads the other images present on the webpages containing the returned images. And [18] attempts to circumvent these limitations by both using a lexical database to generate related queries and translating queries into other languages for use in other regional websites such as http://www.google.es. By crawling the web directly, we avoid the need for such work-arounds and are able to collect a large number of images in a short amount of time.

In this work, we seek to alleviate the burden of data collection for development of face algorithms while at the same time creating an extensive, challenging dataset. We augment existing datasets with images retrieved from the internet. To avoid copyright and privacy concerns, we do not provide the images themselves, neither for existing datasets nor for images retrieved from the internet. Rather, in the user interface, we provide a means to create machine-dependent file paths for images from existing datasets and provide image URLs for those images retrieved from the internet.

TABLE I: Existing Datasets currently included in our Database

Labeled Faces in the Wild http://vis-www.cs.umass.edu/lfw/
MIT-CMU Frontal Faces http://vasc.ri.cmu.edu//idb/html/face/frontal_images/index.html
MIT-CMU Profile Faces http://vasc.ri.cmu.edu//idb/html/face/profile_images/index.html
Caltech Faces 1999 http://www.vision.caltech.edu/html-files/archive.html
MIT-CBCL http://cbcl.mit.edu/software-datasets/heisele/facerecognition-database.html
ORL http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
Caltech 10,000 Web Faces http://www.vision.caltech.edu/Image_Datasets/Caltech_10K_WebFaces/

II. Sources of Images

To compile an extensive, freely available set of images, we leverage two sources: existing, publicly available datasets and images retrieved and indexed from the internet.

A. *Existing Datasets*

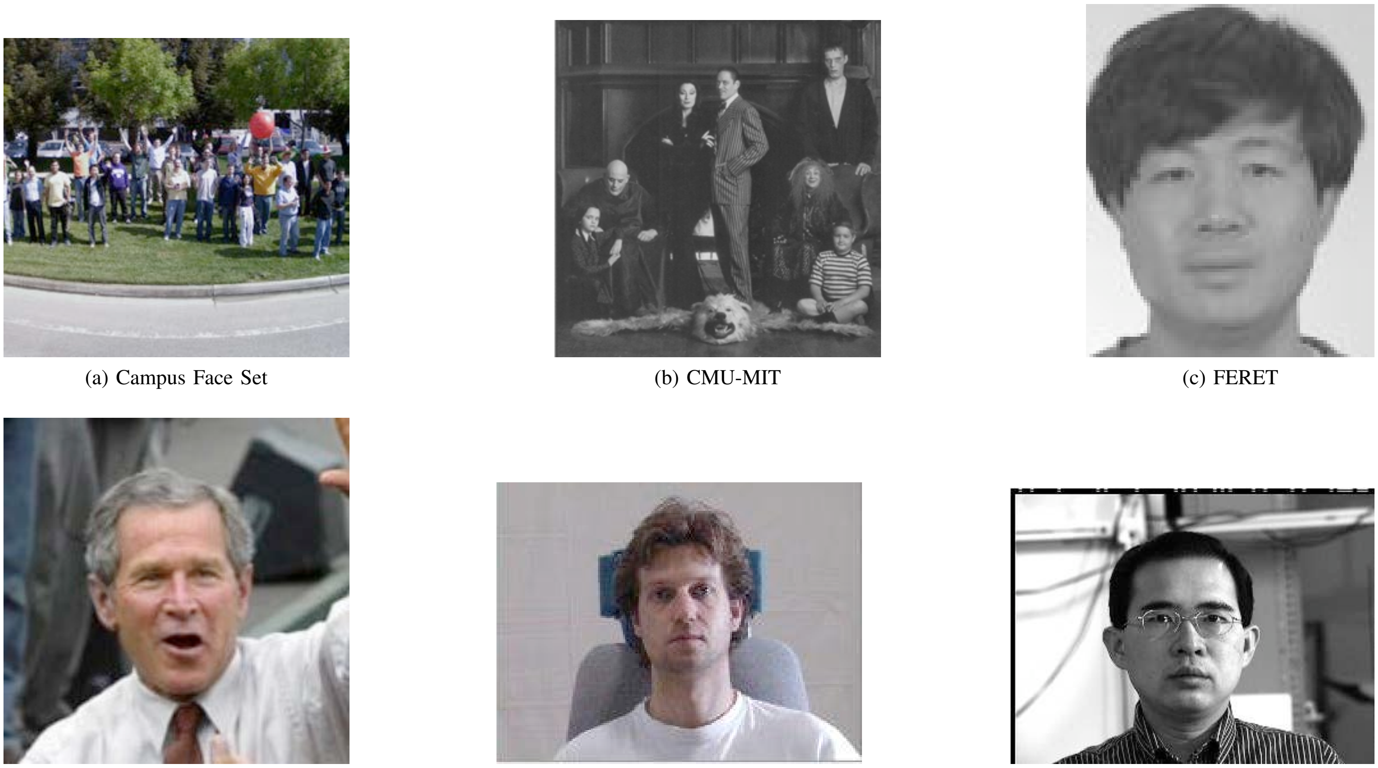
As discussed in the introduction, the computer vision field has expended much effort to compile various datasets for face recognition and detection. These sets of images represent a valuable resource and when publicly available, should be incorporated into our database of face images.

To avoid copyright and privacy concerns, and to also ensure that the original compilers receive due credit, we do not directly provide the images themselves. Instead, links to the original datasets are supplied. All datasets can then be downloaded to a “base” folder on the user’s system. Then, when researchers download the dataset information from our user interface, the file paths can be customized based on the location of the “base” directory and whether the source machine is Unix or Windows based. A list of datasets currently included in our database is included in Table I.

B. *Internet Images*

Clearly the internet is an extensive resource for images. Yet, concerns over copyright and privacy issues often make it difficult to share images retrieved from the internet among different research groups. To alleviate these concerns, we provide URLs to the images, rather than the images themselves.

1) Data Retrieval From the Internet: The crawler used to gather image URLs is based on the Internet Archive’s open-source Heritrix project [19]. It has been modified to extract image URLs along with associated information from the source page. We provide the crawler with a seed list of URLs obtained from a subset of the Open Directory (http://dmoz.org). As the modified Heritrix crawler fetches webpages, it retrieves any image URLs found in the HTML. Along with the image URL, it also stores the time the URL was extracted along with the source page. As noted by Ziou and Bernardi, textual information may also prove useful in an image database [20].



(a) Campus Face Set

(b) CMU-MIT

(c) FERET

(d) LFW

(e) MULTI-PIE

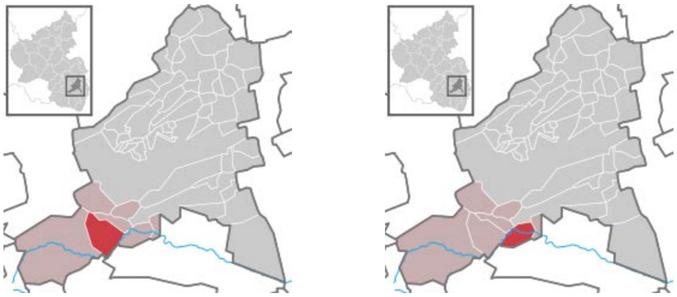
(f) PIE

Fig. 2: **Example Images from Various Datasets**

Thus along with the other image information, we also store the title of the page from which the image was extracted and any alt text included in the image tag.

The images themselves are then separately downloaded and processed. Note that although we download the images in order to mine further information from the images, these downloaded images are not provided directly to outside research groups. Downloading the images separately from the crawler carries two advantages. First, it ensures that errors or delays in downloading images do not slow or cause fatal errors in the crawler itself. Second, by introducing a gap between when the image URL is retrieved and when the image is itself downloaded, we ensure that the image link itself is relatively stable.

After the image has been downloaded, its dimensions and color encoding (grayscale or color) are stored. Then, various processing steps, as described in Section III, are performed on the images to further refine the database.



(a)

(b)

Fig. 3: **False Duplicates** Example of a false duplicate pair

III. Processing of Images

While we seek to store a large variety of images in the database, there are still some undesireable images that should be excluded. Once an image is downloaded, it is then processed to determine whether it will remain in the database.

A. Eliminating Duplicates

A single image may be found on the web on many different sites. To reduce the amount of URLs stored, only a single copy of each image should be stored in the database. Yet, an image may be stored on different servers with different URLs so a priori, there is no simple way to determine whether an image is already in the database.

The naive way to determine the uniqueness of an image is to do a pixel by pixel comparison with images already stored in the database. Alternatively, a hash value can be computed and stored along with the image URL. Ideally, these hashes should have a high probability of being unique and be quick to compute. Many methods have been proposed to hash images. Xiang et al. use a histogram based approach, working on grey scale images [21]. In their Replicated Image Detector System, Chang et al. use features based on Debauchies wavelet coefficients to index images [22]. They then use these indexes

Fig. 4: **Example Duplicates** Examples of duplicate images found in a small run of the crawler.

to find copies of query images. While [22] and [21] take a more global approach in describing images, Ke et al. take a local approach. They use a Difference of Gaussian detector to detect local interest points and use PCA-SIFT to compute feature vectors for interest points. They then index these descriptors and use this information to achieve over 99% accuracy on near-duplicate detection and sub-image retrieval [23].

Many of the schemes discussed above are designed for security purposes (ex. copy detection) and are not necessarily well-suited for a collection of face images. With many near-duplicate detection systems, there is a tradeoff between the number of near-duplicate pairs and the number of false duplicate pairs detected. In this application, we lean towards a method that is more discriminative. That is, we seek to have as few false duplicate pairs as possible at the expense of not excluding some near-duplicate pairs. Indeed, for some applications it is desireable to have near-duplicates included in a dataset. For instance, in face recognition, algorithms must be able to recognize faces under different illumination conditions, poses, and scales. We thus use a histogram based method that is easily implemented, efficient, and discriminative.

To detect duplicates, we compute a hash value for each image. This hash value is then used as a unique key in the database. Images are first resized to 256X256 so each dimension is divisible by high powers of 2. Images are resized using OpenCV with bilinear interpolation. Then, we apply a Gaussian filter to the image. As noted by [21], applying a low pass filter helps to make the hash scheme more robust to artifacts introduced due to image compression.

To compute the hash value, the image is divided into 8 regions. In each region, an 8-bin, normalized histogram of pixel values is computed. Similarly, for each region, an 8-bin, normalized histogram of edge orientations is computed. The edge orientations are computed by applying Scharr filters to calculate the discrete $y-$ and $x-$derivatives and then taking the arctan of the quotient of the two derivatives. Note that a value is included only if its magnitude is above a certain threshold value.

Then, using these histogram values, a 32 bit hash value is constructed based on pairwise comparisons of bins in each histogram. Note that in order to limit the hash value to 32 bits, the bit values for each histogram could not simply be concatenated since this would require $8 * 2 * \binom{8}{2} = 448$ bits. Instead, the hash values computed from each histogram are summed together resulting in a $\binom{8}{2} + \log_2 8 = 32$ bit hash value. While in theory, summing the individual hash values can result in very different pictures having the same hash, in practice this problem is not observed. See Figure 5 for the algorithm. Note that when computing histograms, soft-binning is used. Soft-binning solves the discretization problem of values near histogram bin boundaries being assigned to a single bin. With soft-binning, weighted votes are assigned to each bin based on the distance of the value from adjacent bin centers.

TABLE II: Summary of total number of images retrieved and subsequently removed

Images URLs crawled	31,515	100%
Images Excluded	8,468	26.9%
Duplicates Removed	933	3.0%
Total Images	**22,568**	**70.2%**

See Figure 4 for examples of images that were retrieved multiple times by the crawler and Figure 3 for an example of a pair of false duplicates.

B. Removing Undesired Images

While crawling the web, a large percentage of the retrieved images are logos, image buttons, and other design elements for websites. To remove these unwanted images, we use height and width cutoffs along with so called “stop images.” Together with removing images that are unable to be downloaded, we remove approximately 30% of the image URLs retrieved during the crawl. See Table II for a summary of the number of images retrieved and subsequently excluded during a small sample run of the crawler.

1) Height and Width Cutoffs: Using the approach discussed by Ziou and Kherfi, images with a width below θ_w or height

```
$hash = 0$
for $i = 0$ to $7$ do
  $hist_V$ = computeValueHistogram(REGION($i$))
  $hist_E$ = computeEdgeHistogram(REGION($i$))
  $count = 0$
  for $j = 0$ to $7$ do
    for $k = j + 1$ to $7$ do
      if $hist_V(j) \geq hist_V(k)$ then
        $hash+ = 2^{count}$
      else
        $hash+ = 0$
      end if
      if $hist_E(j) \geq hist_E(k)$ then
        $hash+ = 2^{count}$
      else
        $hash+ = 0$
      end if
      $count + +$
    end for
  end for
end for
```

Fig. 5: **Hash Algorithm**

below θ_h are removed from the database [20].

2) "Stop Images": Similar to the approach taken in Natural Language Processing of ignoring words such as "it" and "the," a list of words that have a high probability of indicating unwanted images was constructed. These include words such as "logo," "footer," and "banner." Images with at least one of these words anywhere in their URL are then passed over by the web crawler.

3) Removing Graphics: Another useful filter would discriminate between graphics and photos. There has been much research into discriminating between large classes of images. Athitsos et al. note that there are certain features that can help to distinguish between photographs and graphics. For instance, graphics tend to have large regions of constant color while photos tend to have more noise. Other observations include that graphics tend to have sharper edges and more saturated colors [24]. Lienhart and Hartmann extended this work, training classifiers to distinguish between photos, photo-like images (ex. ray-traced computer graphics), presentation slides/scientific posters, and cartoons. Using intuitive features such as the prevalent color, orientation of edges, and the total number of colors, they achieve accuracy rates upwards of 99% [25].

Similar to [25] and [24], a variety of features were computed for use in the classfier. First, a normalized color histogram was constructed with 32X32X32 bins and the image was converted to HSV space. Then using the original image, the HSV image, and the color histogram, various features were extracted:

- The bin number and count of the most prevalent color.
- Total number of nonzero bins in the color histogram
- The average hue value.
- Based on a pixel neighbor metric, $d = |r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2|$, the percentage of pixels with $d > 0$
- The percentage of pixels with $d > \theta$, with $\theta = 50$.
- Width to height ratio of the image.

AdaBoost was then used to train a classifer. Real AdaBoost, as implemented in the OpenCV library, was used. As described in [26], the basic idea of AdaBoost is to train T weak classifiers. In the OpenCV implementation, each weak classifier is a decision tree. With each iteration, training examples that have been misclassified by previous classifiers are given more weight. And in the final classifier, each weak classifier is weighted according to its accuracy on the training data. The basic algorithm is given below (adapted from [26]):

1) $D_1(i) = 1/m, i = 1, ..., m$
2) For $t = 1, ..., T$
    a) Find the classifier h_t that minimizes the $D_t(i)$ weighted error:
    $h_t = \text{argmin}_{h_j \in H}(\varepsilon_j)$ where $\varepsilon_j = \sum_{i=1}^{m} D_t(i)$ for $y_i \neq h_j(x_i)$ as long as $\varepsilon_j < .5$.
    b) Set the h_t voting weight:
    $\alpha_t = (1/2) \log((1 - \varepsilon_t)/\varepsilon_t)$.
    c) Update the data point weights:
    $D_{t+1}(i) = [D_t(i) \exp(-\alpha_t y_i h_t(x_i))]/Z_t$, where Z_t is a normalization factor.

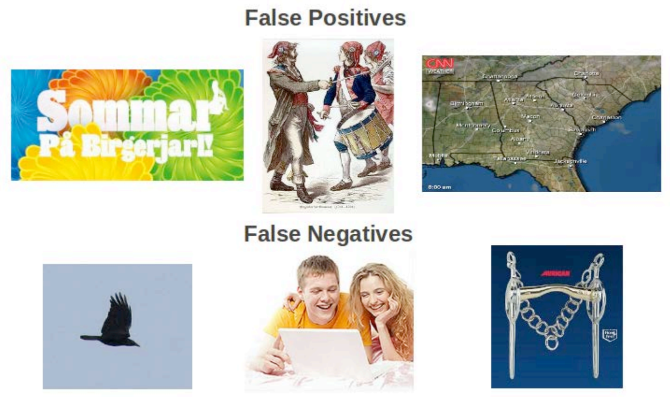


Fig. 6: **Example Missclassified Images** Examples of both false positives and false negatives

The final classfier is given by:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \quad (1)$$

The classifier was trained with approximately 1000 positive examples and 1000 negative examples. These images were retrieved from the web via the crawler. It was then tested on a total of 1500 images. Overall, it achieved a 90.2% true positive and 88.4% true negative detection rate. While these results are lower than that reported in [25], the dataset used to train and test our classifier was retrieved entirely from the web and contained a vast variety of images. In particular, the photos retrieved varied widely. For instance, some photos contained full shots of outdoor or indoor scenes while others were of consumer products taken against a plain background. Some examples of misclassified images are given in Figure 6. In contrast to the varied photos in our dataset, the photos used in [25] were all of nature.

IV. Detecting Faces

By itself, a large database of internet images would prove relatively useless. Thus, faces must be detected in the images, either automatically with existing detectors or manually through human anotations. The use of Viola-Jones based face detectors is discussed in the following subsection while the use of human annotaiton is discussed in Section VI.

A. Viola-Jones Face Filters

OpenCV is an open source project supported by Intel. It provides a large library of functions for use in computer vision research and applications [10]. Included in the OpenCV library is an implemented method of the Viola-Jones face detection algorithm [9]. The Viola-Jones method uses Haar-like features based on sums and differences of rectangular regions in the image and trains a cascade of classifiers based on these features. See Figure 8 for examples of Haar-like features. Since the set of Haar-like features defined by [9] is overcomplete, they use Ada-Boost to select the most dsicriminative features and to train the classifiers. The goal is to quickly eliminate large portions of an image that are very unlikely to contain faces so later classifiers have to search a smaller region of the image. Portions of the image, if any, that make it through the entire cascade of classifiers are then chosen as faces.

OpenCV comes built with five different frontal-face, cascade-based filters. Four are based closely on Viola-Jones, using Ada-Boost with Haar-like features and differ based on the data and variant of Ada-Boost used in training. The fifth also uses a cascade of classifiers trained with Ada-Boost, but instead of Haar-like features, uses local binary patterns (LBP). LBP features were introduced by Ojala et al. as a means to describe local texture patterns [27]. LBP creates a description of a $3X3$ pixel patch by thresholding the outer pixels using the center pixel value. See Figure 7 for an example calculation of an LBP feature.

Note that to help reduce the number of false positives, we run the face detectors only on those images classified as being photos by the algorithm discussed in Section III-B3. See Table III for statistics on the number of faces detected by each filter in a total of 12,521 images classified as being photos by the algorithm in Section III-B3.

TABLE III: Statistics on number of faces detected in a total of 12,521 images by the different cascades. False positive rate was determined by manually classifying 540 of the results returned by each classifer.

Cascade	# Faces Detected	False Positive Rate
frontalface_alt_tree	1,748	24.6%
frontalface_alt	2,376	29.8%
frontalface_alt2	3,069	57.0%
frontalface_default	5,833	47.6%
lbpcascade_frontalface	3,132	65.9%

Note that the statistics on the false positive rate are somewhat misleading since it was simply computed by dividing the number of incorrect faces by the total number of faces returned by the classifier. Yet, in detecting faces the Viola-Jones classifier scans across the entire image at multiple scales, resulting in up to thousands of windows scanned for each image. Thus, the false positive rate would be significantly lower for the cascades if it was computed based on the total number of windows scanned. Further, a majority of the false postives came from a small subset of the images. This was because the classifer was set to detect faces as small as 30X30 pixels. For smaller images, this was a reasonable threshold but for larger images, faces this small are much more unlikely. Thus, a small threshold resulted in a very large number of windows scanned, and hence a proportionally larger number of false positives. Indeed, when a relative threshold of one eighth of the image width and height was set, the false positive rate for the frontal_alt_tree classifier was reduced from 24.6% to 7.7%. Yet, this lower false detection rate also comes with a tradeoff as many of the smaller, harder to detect faces are missed by the classifier. For instance, based on the false positive rate of 24.6% and 7.7% for the frontal_alt_tree cascade with the differently defined thresholds, the former returns 1,318 correct faces while the latter returns 1,119 correct faces. Thus, rather than provide a single filter for each cascade, multiple filters are provided for each cascade, each with different parameters that affect the tradeoff between the number of faces returned and the false detection rate.

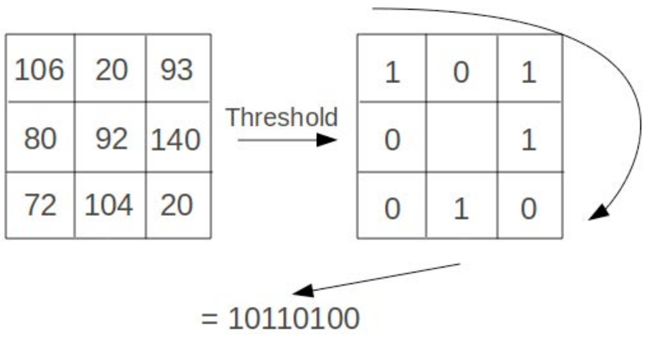


Fig. 7: **LBP** LBP features are generated by thresholding the pixels of a $3X3$ patch with the center pixel

V. User Interface

To provide researchers with an interface to access image data, a site has been designed where researchers can access precompiled datasets or choose to build their own. For custom datasets, users can choose to include existing datasets, such as CMU PIE [6] or LFW [1] and choose to filter internet images using any of the Haar face cascades provided with OpenCV. Researchers can then preview the internet images, excluding undesired ones from their dataset. Preview images are resized to a width of 150 pixels and an appropriately scaled height to maintain the original aspect ratio. The preview images show detected faces so that false positives may be excluded and are also linked to the original image. Further, since small faces may be hard to see in some of the images, users can roll over the images to see a magnified image of just the detected face itself. Additionally, users can choose to label faces not detected by the provided detectors and add the faces to their datasets. Note that due to copyright and privacy concerns, images from existing datasets cannot be previewed in the interface. The custom-made datasets can also be accessed by other researchers although they can only be edited by the original creator.

Once a pre-defined dataset has been chosen or a custom dataset created, researchers can then download the image information as an XML file. For internet images, the URL to the image, rather than the actual image file itself, is provided. For images from existing datasets, researchers specify a base directory where local copies of the datasets are stored on their machine and specify whether their machine is Unix or Windows based. Appropriate file paths are then generated in the XML file. Also included in the XML file is any assocated labeling of the images. Different databases include different standards for labeling faces. For instance, the MIT-CMU dataset includes eye, mouth and nose coordinates while LFW includes only name labels for the images. For each image, we store all available labeling data and in the generated XML file include the labels along with the associated image path or URL.

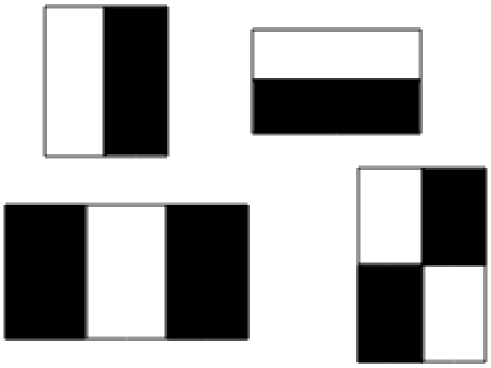


Fig. 8: **Example of Haar-like features** Four different Haar-like features. Black regions are subtracted from white regions

VI. Conclusion and Future Extensions

A time- and resource-intensive task in computer vision is gathering data for use in algorithm design. Indeed, some of the most cited papers in computer vision are of datasets that have been compiled and made publicly available [2], [1], [5], [7], [6]. The wide use of such datasets is evidence that researchers seek both a common benchmark to compare their algorithms to the work of others and also to eliminate the redundant collection of testing and training datasets. And while many publicly available datasets exist, much effort is still wasted among research groups gathering similar data for training and testing.

In this work, we seek to alleviate the burden of data collection in the development of face-related algorithms. We leverage the work of existing face datasets by gathering the existing dataset information into a central location and then expand upon these sets by crawling the web to find additional, and often more challening images. In doing so, we create an extensive, difficult set of face images that can be used for a variety of face-related algorithms.

We have developed our database in such a way to allow for future incremental improvements in both the data collection process and the features provided to the end users. For instance, one area of improvement could be the use of human verification of face images detected on the internet. Indeed, as noted by [13], face detection is far from a solved problem and consequently, it is undesireable to rely solely on existing detectors to gather face images from the web. Yet, labeling data by hand is a tedious and time consuming task. Recently, the Amazon Mechanical Turk service has become a popular way to achieve large amounts of labeling accurately, inexpensively, and quickly. For instance, Kumar el al. used the service to label over 125,000 examples of various human face attributes [28]. And [4] used Amazon Mechanical Turk to label images for use in their ImageNet dataset.

The service allows researchers to post tasks, called "HITS," for human "workers" to complete. The requesters set the payment reward to give workers and can restrict the workers who can complete their tasks based on qualifications such as country of residence. Further, requesters only have to pay for work they consider satisfactory. Amazon Mechanical Turk provides various APIs to programmitcally create new "HITS." Thus, in future versions of the user interface, not only could we further refine pre-defined datasets by submitting "HITS" to Amazon Mechanical Turk, but we could also allow researchers to automatically submit their custom-designed datasets to the service to be verified.

Additionally, we currently only utilize Viola-Jones based face detectors to collect face images form the internet. Yet, a wide variety of other face detectors exist in the literature ([15], [14], [13]) along with commercial detectors ([29], [30]) that could be used to further extend the range of face images included in the database.

To summarize, we provide an extensive database of face images that is freely available for researchers to use for face related algorithms. By providing such a large store of images in a central location, we aim to reduce the collective time and resources devoted to data collection among the computer vision field.

References

[1] G. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," *University of Massachusetts, Amherst, Technical Report*, vol. 57, no. 2, pp. 07–49, 2007.

[2] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer Vision and Image Understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[3] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[4] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: a large-scale hierarchical image database," 2009.

[5] P. Phillips, H. Moon, P. Rauss, and S. Rizvi, "The FERET evaluation methodology for face-recognition algorithms," in *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997. Proceedings.*, 1997, pp. 137–143.

[6] T. Sim, S. Baker, and M. Bsat, "The CMU pose, illumination, and expression database," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1615–1618, 2003.

[7] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "The CMU multi-pose, illumination, and expression (Multi-PIE) face database," Technical report, Robotics Institute, Carnegie Mellon University, 2007. TR-07-08, Tech. Rep.

[8] T. Berg, A. Berg, J. Edwards, M. Maire, R. White, Y. Teh, E. Learned-Miller, and D. Forsyth, "Names and faces in the news," 2004.

[9] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple," in *Proc. IEEE CVPR 2001*. Citeseer.

[10] G. Bradski, "The OpenCV Library–An opensource library for processing image data," *Dr. Dobbs Journal*, pp. 120–125, 2000.

[11] T. Kanade, S. Baluja, and H. Rowley, "Rotation Invariant Neural Network-Based Face Detection," 1997.

[12] M. Fink, R. Fergus, and A. Angelova, "Caltech 10, 000 web faces," http://www.vision.caltech.edu/Image_Datasets/Caltech_10K_Web Faces/.

[13] A. Frome, G. Cheung, A. Abdulkader, M. Zennaro, B. Wu, A. Bissacco, H. Adam, H. Neven, and L. Vincent, "Large-scale Privacy Protection in Google Street View," *California, EUA*, 2009.

[14] L. Zhang, R. Chu, S. Xiang, S. Liao, and S. Li, "Face detection based on multi-block lbp representation," *Advances in Biometrics*, pp. 11–18, 2007.

[15] C. Huang, H. Ai, Y. Li, and S. Lao, "High-performance rotation invariant multiview face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 671–686, 2007.

[16] N. Kumar, P. Belhumeur, and S. Nayar, "FaceTracer: A search engine for large collections of images with faces," in *Proceedings of the 10th European Conference on Computer Vision: Part IV*. Citeseer, 2008, p. 353.

[17] F. Schroff, A. Criminisi, and A. Zisserman, "Harvesting image databases from the web," 2007.

[18] B. Collins, J. Deng, K. Li, and L. Fei-Fei, "Towards scalable dataset construction: An active learning approach," *Computer Vision–ECCV 2008*, pp. 86–98, 2008.

[19] J. E. Halse, G. Mohr, K. Sigurdsson, M. Stack, and P. Jack. Heritrix user manual. Internet Archive. [Online]. Available: http://crawler.archive.org/articles/developer_manual/index.html

[20] M. Kherfi, D. Ziou, and A. Bernardi, "Image retrieval from the world wide web: Issues, techniques, and systems," *ACM Computing Surveys (CSUR)*, vol. 36, no. 1, pp. 35–67, 2004.

[21] S. Xiang, H. Kim, and J. Huang, "Histogram-based image hashing scheme robust against geometric deformations," in *Proceedings of the 9th workshop on Multimedia & security*. ACM, 2007, p. 128.

[22] E. Chang, J. Wang, C. Li, and G. Wiederhold, "RIME: A replicated image detector for the world-wide web," in *Proc. of SPIE Symposium of Voice, Video, and Data Communications*, vol. 3527. Citeseer, 1998, pp. 58–67.

[23] Y. Ke, R. Sukthankar, and L. Huston, "Efficient near-duplicate detection and sub-image retrieval," in *ACM Multimedia*, vol. 2004. Citeseer, 2004.

[24] V. Athitsos, M. Swain, and C. Frankel, "Distinguishing photographs and graphics on the world wide web," in *IEEE Workshop on Content-Based Access of Image and Video Libraries, 1997. Proceedings*, 1997, pp. 10–17.

[25] A. Hartmann, "Classifying images on the web automatically," *Journal of Electronic Imaging*, vol. 11, no. 4, pp. 1–0, 2002.

[26] G. Bradski and A. Kaehler, *Learning opencv*. O'Reilly, 2008.

[27] T. Ojala, M. Pietikainen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[28] N. Kumar, A. Berg, P. Belhumeur, and S. Nayar, "Attribute and simile classifiers for face verification," in *IEEE International Conference on Computer Vision (ICCV)*. Citeseer, 2009.

[29] "Omron: Okao vision," http://www.omron.com/rd/vision/01.html, 2008.

[30] M. Nechyba and H. Schneiderman, "PittPatt face detection and tracking for the CLEAR 2006 evaluation," *Multimodal Technologies for Perception of Humans*, pp. 161–170, 2007.

Creating a Multilingual Lexical Resource

Richard Seliga
School of Computer Science
University of Colorado, Colorado Springs
Colorado Springs, Colorado 80920
rseliga@uccs.edu

Abstract - **In this paper we describe how to create a multilingual lexical resource using corpora. This resource will be available to researchers and the public using a web interface. The input data includes millions of words in different languages. These words will of course have to go through some preprocessing before finally being uploaded to a MySQL database. Keeping the data indexed is crucial and will allow easy access and quick searches. The searchable information will include the number of occurrences of a particular word, significant bigrams and trigrams that include this word, significant right or left neighbors, parts-of-speech for words, relationship graph among the words and example sentences where this word appears. This can all be done creating four to five tables for each language, which we will be using. Later on a user based dictionary will be created to add another aspect to this linguistic recourse. We will create a different database for each language as it seems the most convenient. The data will stress languages like Slovak and Assamese, but will also include other languages like Czech, English, Polish, Bengali and Kannada. The reason for creating this website is creating a resource for languages that are not as common to find recourses for on the web.**

I. Introduction

This project is intended to create a large linguistic library of words, their uses and their definitions for those who speak an Indic and Slavic languages, but our initial focus will include languages like Assamese and Slovak. A website will be created to allow the user to research relationships among words in terms of occurrences, bigram or trigram frequencies, and significant right or left neighbors; part of speech for words and usages of the words in different part of speech; and will also allow the user to add definitions to a user based dictionary. Another feature that will be developed will include the ability to search for root words. After all this is finished the website should show about two pages worth of information for every word entered. Part of speech tagging will be a challenge and we will either use an existing POS tagger or work on to develop a new one. All the information for these different options will be extracted from the corpora that are free around the web. This will be a great resource for those who want to:

- have access search engine for lexical resources
- get statistical information about your query word
- have access to lexical information on uncommon languages like Slovak and Assamese.

II. Related Work

A project has been started by the Leipzig University, Computer Science Institute in the turn of the millennium and their website now receives more than 170,000 monthly visits. They have come up in their proposal with a table that shows how many words are in their main corpora (see Table 1). [4][1] Obviously with the languages we will be using, we will not have as many courses, but using Wikipedia dumps and free corpora we will works ourselves up to a nice number. Biemamn wanted to create a flexible website that allows people from around the world to research relationships among words. He also created a great standard of comparison for new websites.

Table 1

	German	*English*	*Italian*	*Korean*
Word Tokens	500 Mill.	260 Mill.	140 Mill.	38 Mill.
Sentences	36 Mill.	13 Mill.	9 Mill.	2.3 Mill.
Word Types	9 Mill.	1.2 Mill.	0.8 Mill.	3.8 Mill.

Another future addition might include graphing the relationships of word. Hatzigeorgiu created a project that displays the data about this. In his paper he describes how world length and occurrences show up on a graph. When he mapped out word length against the number of occurrences he came out with some pretty interesting results. It will be fascinating to see how it is mapped out in Slovak and Assamese, and other Indic and Slavic languages. This is another thing that might be added to our website later on. [3]

III. Data Recourses

A. Data Sources

The main and only source of the data is free corpora which are available on the web or have been developed by universities. These collections of text provide anywhere from 2 million to 36 million sentences in each language. The corpora that we will be using include the American National Corpus and Slovak National Corpus for the beginning and later on expand the languages to mostly Indic and Slavic languages, like Assamese, Bengali, Kannada and Czech, Polish. Another option for more text to work with will include extracting Wikipedia articles in their respective languages. Here is some of the data we obtained from the three languages so far (see table 2).

Table 2

	English	*Slovak*	*Assamese*
Word Tokens	11.1 Mill.	17.7 Mill.	2.431 Mill.
Sentences	1.8 Mill.	2 Mill.	.
Word Types	0.23 Mill.	0.94 Mill.	0.21 Mill.

The American National Corpus[1] contains over 6000 documents while the Emille[2] corpus, developed by the University of Lancaster one is much smaller. The 17.7 million words in the Slovak language were obtained from a XML Wikipedia dump. We used this Wikipedia dump because the Slovak national corpus is not as accessible as we first thought.

B. Text preprocessing

All the text processing in our project will be done using Perl, since it has great capabilities with word processing. In this section we will describe the steps to construct a text database

1. Create a two dimensional array of strings that includes the sentence and their respectable words.
2. Strip all punctuation from the text document.

Each word will have the index of what sentence it is in and the index of the position in the sentence. This will eliminate getting inaccurate data. This is necessarily because the end of a sentence and a beginning of another is not a bigram. The next step will involve counting the number of accurateness of each word in the documents and sending this data into a database of unigrams which will include the primary key of the unigrams, the spelling of the word and how many times it has occurred in the corpus. The index will provide us with an easy way to connect two or three words. The connecting of words will be based on the significant right or left neighbors of the word. This will create collocation which is the occurrence of two or more words within a sentence or a document. We will keep count of this data and display it on request of the user. By indexing the sentences, we will create a table including the id of the sentence and merge it with the unigram table to display what unigrams appear in what sentences. One of the other tables we will create is a root table, which will display the root word.

A. Creating the Database

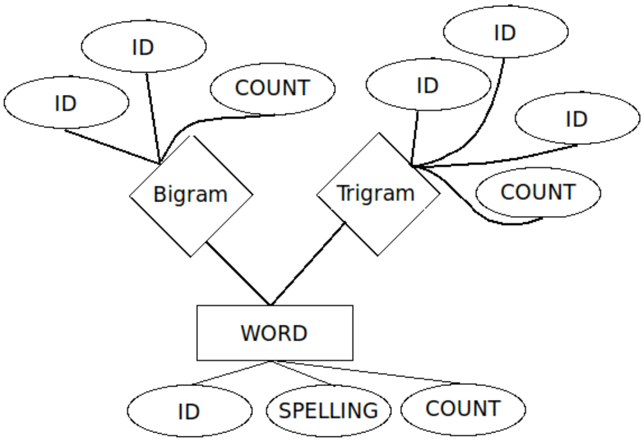
The database will be one of the most crucial parts of this project. It will supply our users with quick responses to their queries which will be accessed through a web interface. The main table will be the unigram table. The unigram table will include the id, spelling and the count. To get bi-grams and trigrams all that will be done is connect two and three word ids. The sentences will be in another table and the only relation it will have to the unigram table is that it receives the data from the same document collection.

Most available sentence splitters are not very good and sometimes cause errors with one word sentences. This problem will be avoided by querying only sentences between a certain character lengths. Indexing the data will make it extremely quick to access, and will keep the users happy. The entity relationship diagram can be seen for our data can be seen on figure 1 and some of the most popular unigrams in the English language can be seen it table 3.

Table 3

ID	*SPELLING*	*COUNT*	*ID*	*SPELLING*	*COUNT*
1	the	721015	41	is	121803
21	of	385945	33	for	115616
49	and	298341	159	with	85187
11	to	268803	213	as	76022
35	in	241763	71	on	70113
4	a	235332	7	by	67411
40	that	132009	247	was	64532

Figure 1



IV. Future Work

A. POS Tagging

Using a part of speech tagging algorithm we will create another resource for our users. It will show the word queried by the user and see what POS it is and display it. This will be different for the Slovak language as it has a much more complex grammar. We will start with Slovak, but later extend the work to other Slavic languages and one or more Indic languages. The biggest problem of tagging the Slovak language is that the tools for the Slovak language are underdeveloped since Slovakia claimed its independence from Czechoslovakia in 1993 and most of the people that did research did it in Czech. On the positive side, the Slovak language is very similar to Czech and a POS tagger has been created named ajka for the Czech language.

1) Czech POS Tagging:

For illustration, let’s assume word-form zdi (walls). One of the morphological annotations corresponds to the genitive

[1] American National Corpus {http://www.americannationalcorpus.org/}
[2] The Emille Corpus{www.ling.lancs.ac.uk/corplang/emille/}

singular for feminine nouns, other to the dative, vocative and locative singular, or nominative and accusative plural of the same word. The other corresponds to the imperative of singular of the verb and so on. Each morphological category (case, gender, number...) may take a set of possible values (gender - masculine animate, masculine inanimate, neuter, and feminine). The morphological annotations of a word form represent the combinations of morphological categories for the particular part of speech classes. [7] This creates a lot of different variations and will make the tagging extremely tough.

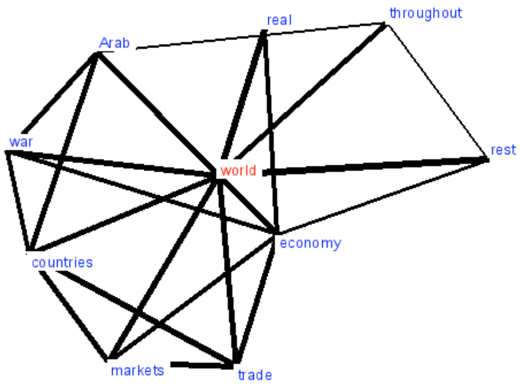
B. *User Input Definitions*

One of the later on additions to this website will include the ability for the user to enter their own definition of the word. Then the definition will be run through a Spam filter and if it successfully runs through it then the definition will be added to the website automatically and be viable by the users. One of the options that we see is using a summarizing posts algorithm to summarize the contents of the user entry and combine this to create some very reliable definitions.

C. *Word Relation Graphs*

Another future addition to this website will include a diagram between different words and their most common neighbors. Even though this seems simple, connecting the words neighbors and their neighbors' neighbors all within one graph will become quite complex and difficult to display properly. Figure 3 shows the graph located on wortzchatz[4]. We will try to make our relationship model a little different this. Also the bolder the line, the more occurrences are related to this subject.

Figure 2



D. *Other Minor Additions*

Some of the minor additions that would be added in the future is that every word displayed on the site would beside it have its count in the corpus and if clicked would display the definition of the word.

V. EXPERIMENTS

Some of the experiments so far have included testing of a website that connects to a database and searches for the most common bigrams and trigrams. The results posted look very promising. The text documents tested included punctuation marks, random lines and double spaces. After testing the same code on the Slovak and Assamese language we had to tweak our code a little to use different encoding. Following this we had 3 databases with all the information needed. Subsequent to testing, we tested some different way to display sentences and we figured out that most sentences that are important are between 50 to 120 characters and this helped us displaying some very good data.

VI. APPROACH TO SOLVING PROBLEMS

Our approach to solving the problems will involve using the scripting language Perl. It is very good with text manipulation and that is what this project needs. Using Perl we first figured out how to count the letter frequencies and word frequencies. Currently we are working on bigrams and trigrams matching by finding out the word that comes before or after the word entered by the user. We will achieve the frequencies by checking the database during insertion. After this is finished we will start working on a basic sentence splitter that will allow us to create a database of sentences. Part of speech tagging will be the final part of this project. After all this has been tested and is working we will implement the different languages and start working on the website. What we did first was made our document available for extraction of information by deleting out some of the punctuation and other unnecessary content. When this was finished, the extraction was quite simple using a *foreach* statement. What we did was put my text document into an array of words and later on used a while loop to look for the appropriate word, while keeping a counter of how many words there are before the current word. When the word was finally found we just looked for the word that had an index of one higher than the counter and one lower than the counter. The solution for the parts of speech tagging will be very interesting to figure out.

VII. SLOVAK LANGUAGE

A. *Slovak Morphology*

Like most other Slavic languages, and contrary to English or German. Slovak is an inflected language. Basically, there are three major types of word-forming processes inflection, derivation, and compounding. Inflection refers to the systematic modification of a stem by means of prefixes and suffixes. Inflected forms express morphological distinctions like case or number, but do not change meaning or POS. In contrast, the process of derivation usually causes change in meaning and often changes of POS. Compounding deals with the process of merging several word bases to form a new word.

B. *Slovak Data Collected*

Some of the data collected from the Wikipedia dump has very good size and quality. In table 4 you can see the most common words in our Slovak database. Since the sentence structure in Slovak is very similar to the one in English we used our English splitter and the results looked very good. The only problem that we ran into was that our Wikipedia dump included a lot of Wikipedia titles and this counted as sentences. We avoided displaying these by only selecting the sentences that had above a certain character count. This proved to be a solution to that problem and after testing some words the data displayed proved to be good sentences.

Table 4

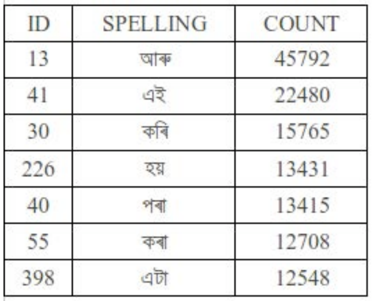
ID	*SPELLING*	*COUNT*
119	v	575566
136	a	517223
154	na	295515
132	sa	286466
116	je	257444
120	roku	109615
190	aj	99273

VIII. ASSAMESE LANGUAGE

A. Assamese Data Collected

We used an Assamese corpus which was not huge in size but it had a lot of quality text. It was all pre parsed and was a freeze to use. In table 5 you can see the most common words in our Assamese database. Since the Assamese language uses Eastern Nagari Script we had to change our code to a different encoding to handle it. The data displayed looked great.

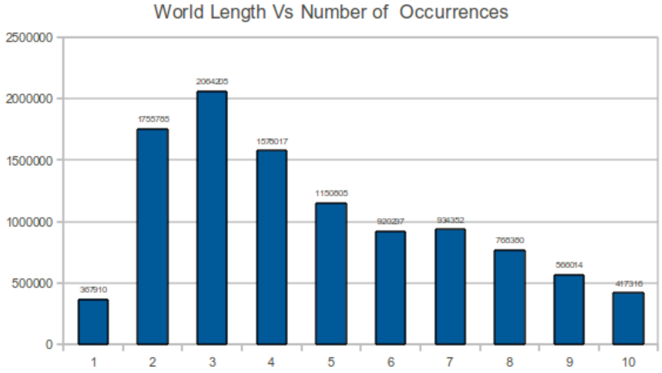
Table 5



ID	SPELLING	COUNT
13	আৰু	45792
41	এই	22480
30	কৰি	15765
226	হয়	13431
40	পৰা	13415
55	কৰা	12708
398	এটা	12548

IX. WORD LENGTH VS. NUMBER OF OCCURRENCES

A. English



X. CONCLUSION AND FUTURE ADDITIONS

Part of speech tagging is going to be one of the more challenging topics addressed and implemented into the website. Horak, the developer of this tagger uses a similar tool that has been used for the Czech language [5]. It's called AJKA and has been developed by Sedlacek. He describes three major parts of word-forming processes: inflection, derivation, and compounding. Inflection refers to the systematic modification of a stem by means of prefixes and suffixes. Inflected forms express morphological distinctions like case or number, but do not change meaning or POS. In contrast, the process of derivation usually causes change in meaning and often change of POS[6]. Compounding all of these together causes problems that are unlike anything in the English language.

In conclusion getting this project will be very challenging and time consuming to work properly. This project would be a good challenge for an intermediate programmer. The final product is something that I am very much looking forward too as it will be extremely satisfying. The final thoughts on this project and some later additions would include an on screen keyboard pop up that would allow users in India, where the amount of characters is so extreme it forces the population to only type in English causing only ten percent of the population to have access to computers. Another addition would be a dictionary that would allow the users to enter the definition in their own language through this on screen keyboard and make the site a very good resource that is user based.

REFERENCES

[1] U. Quasthoff and M. Richter and C. Biemann, Corpus Portal for Search in Monolingual Corpora, Leipzig, Germany: Augustusplatz 11, 04109, 2006.

[2] L. Egghe, The distribution of N-grams, Akadmiai Kiad, Budapest, 2000.

[3] N Hatzigeorgiu, G Mikros, and G Carayannis, Word Length, Word Frequencies and Zipfs Law in the Greek Language, Maroussi, Greece, 2001.

[4] U. Quasthoff and M. Richter and C. Biemann, Language- Independent Methods for Compiling Monolingual Lexical Data, Leipzig, Germany: Augustusplatz 11, 04109, 2004.

[5] A. Hork, L. Gianitsov, M. imkov, M. motlk, and R. Garabk, Slovak National Corpus, Ludovt tr Institute of Linguistics, Slovak Academyof Sciences Bratislava, Slovakia, 2004.

[6] Radek Sedlacek and Pavel Smrz, LA New Czech Morphological Analyser ajka, Faculty of Informatics, Masaryk University Brno Bota nicka 68a, 602 00 Brno, Czech Republic, 2001.

[7] Jan Hajic and Barbora Hladka, Czech Language Processing – PoS Tagging, Institute of Formal and Applied Linguistics Charles University Malostransk nm. 25 118 00 Prague, Czech Republic

[8] Benot Sagot, Automatic Acquisition of a Slovak Lexicon from a Raw Corpus, INRIA-Rocquencourt, Projet Atoll, Domaine de Voluceau, Rocquencourt B.P. 105 78 153 Le Chesnay Cedex, France

Extraction and Visualization of Temporal Information and Related Named Entities from Wikipedia

Daryl Woodward
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy Colorado Springs, CO 80918
daryl.woodward@gmail.com

Abstract—**This paper addresses our process in generating a tool that extracts named entities and events from a document and visualizes them in ways beneficial to someone learning about the topic. The ultimate goal is to present a user with many of the key events and their associated people, places, and organizations within a document that will quickly give users an idea of the contents of an article. For testing, we use a set of historical Wikipedia articles which focus on topics such as the American Civil War. These articles have high occurrences of all types of named entities along with many events with clearly defined time spans. For initial named entity extraction, we incorporate the Stanford NLP CRF into our project. In recognizing location names in this subject area, it only achieves an f-measure of 57.2%. The list of locations is geocoded through Google Geocoder and will be disambiguated through a tree structure in the future. A final f-measure of 79.1% is determined which represents the precision and accuracy of our package in successfully grounding the extracted locations. The grounded locations are then grouped with other named entities related to an event through sentence-level association. Visualization is currently done through Google Maps and the Timeline SIMILE project developed at MIT. We plan to add the capability to geospatially and temporally refine article searches in Wikipedia and make our tool usable on other online corpora.**

I. Introduction

The Internet has given mankind an efficient method of sharing information. As the amount of data increases, we need ways to express it effectively, especially for learning. Visualization can sometimes offer a level of understanding not inherent in reading the text alone. By generating a tool that extracts information open to various types of visualization, we facilitate the addition of future features for our tool and new tools all together. Over the past couple of decades, the Internet has gained a foothold in various aspects of people's every day lives all over the world. One of the most influential features of the Internet is the ability to easily collaborate in generating information. A perfect example was the creation of the wiki. A wiki offers an ideal environment for the sharing of information while allowing for a form of peer review that is not present in many privately operated websites. The largest wiki that exists today is Wikipedia. Only recently created in 2001, the English Wikipedia has already grown to over 3.3 million articles [1]. Wikipedia is only one of many corpora that can be mined for knowledge and displayed in concise form. Some examples of other corpora reside in the genres of news articles, journals, books, blogs, etc. It should be noted however, that although such online sources can offer a great deal of information, readers often need quick, decisive information from an article without reading the whole thing. Thus begins the motivation of our work.

II. Motivation

Our work begins with a focus on extracting knowledge from Wikipedia. After the tool has been fully evaluated on this corpus, testing will be extended to archived news articles and then RSS (Really Simple Syndication) feeds. Although Wikipedia offers a basic article structure and ways for authors to relate articles together, tools have been and should continue to be created to automate the extraction of important information from these articles. With over a thousand articles being created per day[2], Wikipedia has the potential to be used in many educational environments. One task that needs to be implemented is the creation of a knowledge base where key facts can be identified extremely efficiently, especially from different online sources. If various encyclopedias are analyzed, much of the same information should be pulled out by this sort of tool. Thus, a researcher could easily verify or disprove information from a single source. This task however, has various obstacles associated with it.

III. Background Information and Related Work

In regard to querying and extracting knowledge from Wikipedia, Auer and Lehmann demonstrated an efficient algorithm for categorizing articles and extracting information from Wikimedia templates [1]. Such information may be effective in general queries but does not extract deep enough content to be applied to the visualization in our work. It may however, prove to be useful in the expansion of search features in the future. The idea of extracting information from the predefined Wikipedia data structure may be applied to this work but will be more difficult when templates and tags do not exist in the documents being processed (in other corpora). Similarly, Mihalcea and Csomai developed a keyword extraction algorithm

[1] http://en.wikipedia.org/wiki/Wikipedia

[2] http://en.wikipedia.org/wiki/Wiki#History

to identify important words from within a document and link them to their respective Wikipedia pages [2]. This can aid in future work as it can help identify pages related to the target document that should be processed alongside it to get a more complete set of important information.

A variety of approaches have been applied to NER since MUC-6 in 1995[3] including Hidden Markov Models, Conditional Random Fields, Maximum Entropy models, Neural Networks, and Support Vector Machines (SVM). The extraction of named entities continues to invite new methods, tools, and publications. Basic named entity extraction is performed in [3] and [4]. Both of these works focus on the extraction/visualization of named entities from RSS feeds.

Specifically, Chen et al. performs Named Entity Recognition with a "regularized maximum entropy classifier with Viterbi decoding" [4] and achieves an f-measures of over 88% in regard to geospatial entities. Our goal is to achieve recall in the same area after initial extraction, but with people and organizations as well. Precision can be sacrified in the realm of geospatial entities as many of the inaccuracies will be weeded out when the places are geocoded.

[3] uses an interesting disambiguation process in their geospatial resolution process. Nearby named entities are used to disambiguate more specific places. An example is a state being used to determine exactly which city is being referenced. In addition,NE's previously extracted in the document are also used to reinforce the score of a particular instance of a place name. For example, if a particular region in the U.S. has been previously referenced and a new named entity arises that could be within that region or in Europe, it will be weighted more towards being the instance within that region in the U.S. The basic design of our GUI is based off of the GUI referenced in [4].

SVMs have shown significant promise for the task of NER. [5] demonstrated an SVM that achieved an f-measure of 0.954 for location entities in Wikipedia articles, and an f-measure of 0.884 across all NE classes. Although research into text classification and NER has found that SVMs provide good performance on NER tasks, HMMs can produce similar results with minimal training.

Hidden Markov Models (HMMs) have also shown excellent results. [6] demonstrated that a Character-level HMM can identify both English and German named entities with an f-measure of 0.899 and 0.735 for location entities in testing data, respectively. [7] evaluated a HMM and HMM-based chunk tagger on the MUC-6 and MUC-7 English NE tasks, achieving f-measures of 0.966 and 0.941, respectively.

The approach we took however is the implementation of the Conditional Random Field provided by the Stanford NLP Group (Covered in more depth in Approach-Tools). We chose this tool because it has satisfactory performance and accuracy. Achieving results close to 90%, the CRF is publicly available, already trained, easily integrated into the Geografikos package, and currently seems to be quite efficient.

IV. Approach

Our focus is on extracting information from a corpus of historical Wikipedia articles. These have high occurrences of dates, times, people, places, etc. that make up events. Such information can be very valuable in evaluating historical topics as these can often be lengthy, sometimes dry articles. We have identified six major steps in representing this sort of information:

1) Extract temporal information which identifies when an event occurs
2) Tag the locations (in regard to the article) of these events
3) Extract named entities and relate them to their respective events
4) Save information back into a database
5) Combine the information for visualization on a map
6) List events and associated entities, perhaps incorporating associated pictures

The overall goal is to generate a two part GUI. The first part emphasizes the visualization of locations. It consists of a map generated by the Google Maps Javascript API[4]. To the side is a list of locations shown on the maps, each clickable to center and zoom the map on that location. Individual markers on the map are clickable at which point an infobox will be displayed with relevant information about the location, including a list of events that occurred there. Clicking an event should open an infobox about the event in the second part of the GUI. This second part is a timeline that is displayed with what we have identified as the most important events of an article. Each event is clickable to display text extracts from the article and links to zoom in on associated locations. A sliding bar will separate the two and the bar itself will have adjustable endpoints. This bar can then be moved from left to right along the overall timespan of all events mentioned in the article. The bar itself represents the time period the user is interested in. If the starting point (left end) of the bar is adjusted all the way to the left (aligning with the least recent event mentioned in the article) and the right end of the bar aligns with the most recent time mentioned in the article, then all events will be visualized on the timeline and their associated locations will all be plotted on the map. This is shown in Figure 1.

A. Fusion Table Approach

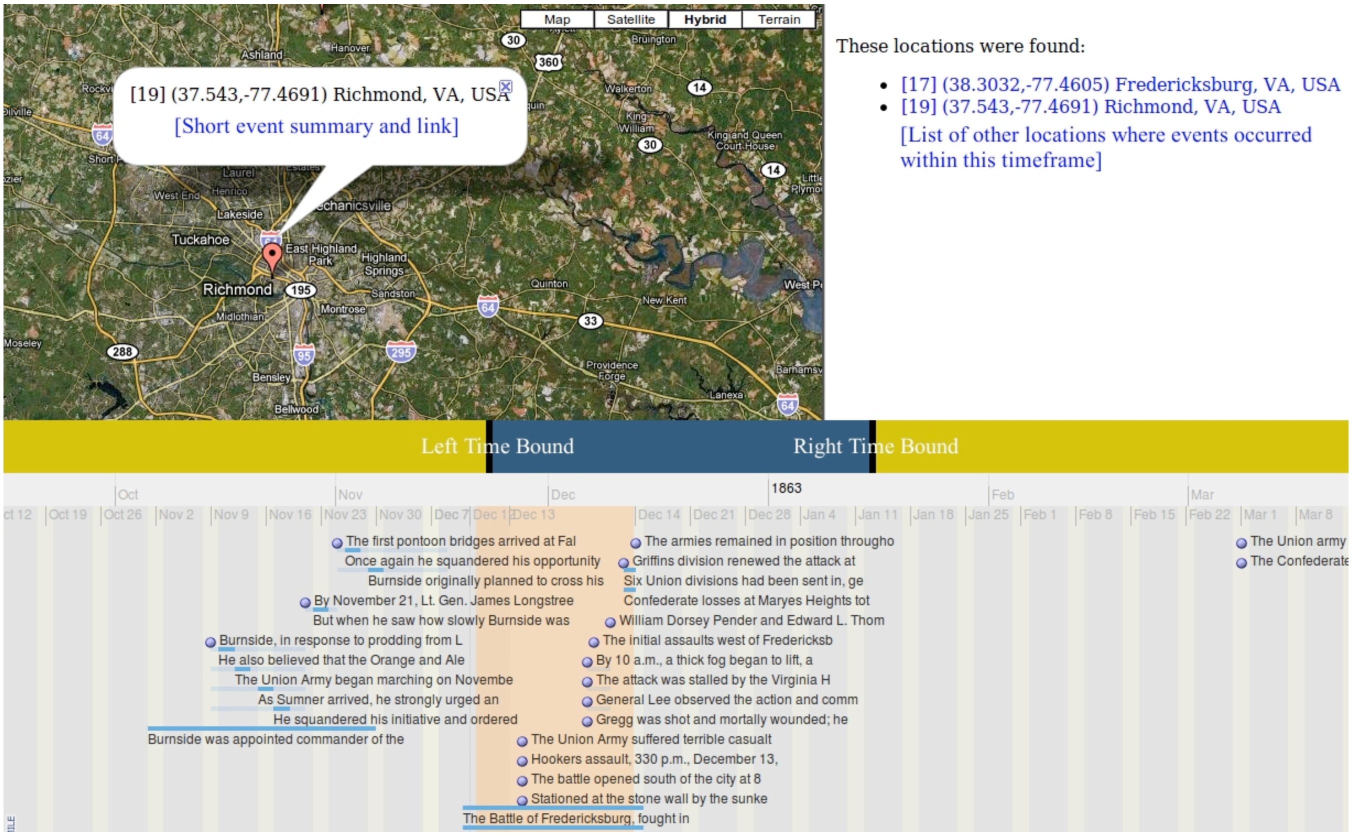
1) Introduction: Here, we discuss a useful tool for visualizing geospatial data but not efficient enough to include in our final implementation. Fusion Tables is a Google Labs project available for anyone with a Google account[5]. Fusion Tables are online database tables which can be queried and updated through simple POST and GET commands over the internet. Google has an API called Google Data Protocal (GData) which "is a REST-inspired technology for reading, writing, and modifying information on the web." [6]. This eases the online interaction between a developer's program and Google's online applications. For this project, we initially attempted to use Fusion Tables due to its simple integration into Google Maps. Google Maps allows the addition of an "overlay layer"

[3] http://cs.nyu.edu/cs/faculty/grishman/muc6.html

[4] urlhttp://code.google.com/apis/maps/documentation/javascript/

[5] http://tables.googlelabs.com

[6] http://code.google.com/apis/gdata/



[19] (37.543,-77.4691) Richmond, VA, USA
[Short event summary and link]

These locations were found:

- [17] (38.3032,-77.4605) Fredericksburg, VA, USA
- [19] (37.543,-77.4691) Richmond, VA, USA

[List of other locations where events occurred within this timeframe]

The first pontoon bridges arrived at Fal
Once again he squandered his opportunity
Burnside originally planned to cross his
By November 21, Lt. Gen. James Longstree
But when he saw how slowly Burnside was
Burnside, in response to prodding from L
He also believed that the Orange and Ale
The Union Army began marching on Novembe
As Sumner arrived, he strongly urged an
He squandered his initiative and ordered
Burnside was appointed commander of the
The armies remained in position througho
Griffins division renewed the attack at
Six Union divisions had been sent in, ge
Confederate losses at Maryes Heights tot
William Dorsey Pender and Edward L. Thom
The initial assaults west of Fredericksb
By 10 a.m., a thick fog began to lift, a
The attack was stalled by the Virginia H
General Lee observed the action and comm
Gregg was shot and mortally wounded; he
The Union Army suffered terrible casualt
Hookers assault, 330 p.m., December 13,
The battle opened south of the city at 8
Stationed at the stone wall by the sunke
The Battle of Fredericksburg, fought in
The Union army
The Confederate

Fig. 1. Basic Format of GUI

that automatically pulls geospatial data straight from a Fusion Table. Our first attempt at visualizing information was by storing data both offline in a MySQL database and online in these Fusion Tables. The data stored online was much more minimal and consisted only of the geospatial information associated with locations. Two online tables were created similar to their offline counterparts. These are listed below with their associated columns: With this format we can do searches by article or location. One example is to retrieve all locations mentioned in one article by matching:

Pages Table	
page_id	loc_ids

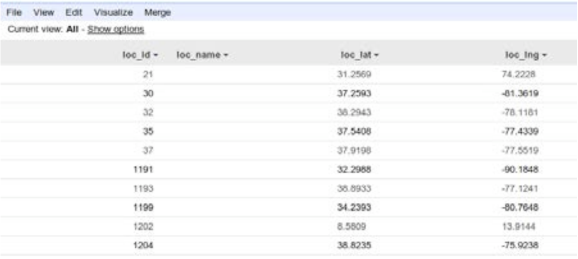
Locations Table								
loc_id	loc_name	loc_lat	loc_lng	loc_street	loc_city	loc_state	loc_country	sent_id

```
pages.id = query.id and page.loc_ids = locations.loc_id
```

Fusion Tables also offers various sharing options where one can not only share raw data but visualizations just as easily. For example we can choose to identify the latitude and longitude columns as a pair that make up a location. Then we can just click on a Map button to map all the places or embed a map in our own application using GData. A user can also easily join tables together. Part of the locations table is shown in Figure 2.

B. Implementation

Figure 3 shows a map of all locations found in the Battle of Fredericksburg and Nickel Grass articles. This is just a simple example where all locations in the table are added to the map as a Fusion Table Layer. Without any additional code, the map will automatically include the infoboxes that display any other information stores in the table upon clicking a marker. Although the use of Fusion Tables is convenient, the speed of processing was quite slow. In addition to initially mining an article for geospatial entities, about a half second to a full second was taken per query or update to the online tables. This time was significantly decreased by multithreading this portion of processing. One thread managed the pages table and the other managed the locations table. Using more than one thread to access one table generated errors, even when short delays were added in. Ultimately, we decided not to continue using Fusion Tables as this added minutes to the processing time for each article but it could be useful for smaller applications.



Current view: All - Show options

loc_id	loc_name	loc_lat	loc_lng
21		31.2569	74.2228
30		37.2593	-81.3619
32		38.2943	-78.1181
35		37.5408	-77.4339
37		37.9198	-77.5519
1191		32.2988	-90.1848
1193		38.8933	-77.1241
1199		34.2393	-80.7648
1202		8.5809	13.9144
1204		38.8235	-75.9238

Fig. 2. Locations Fusion Table

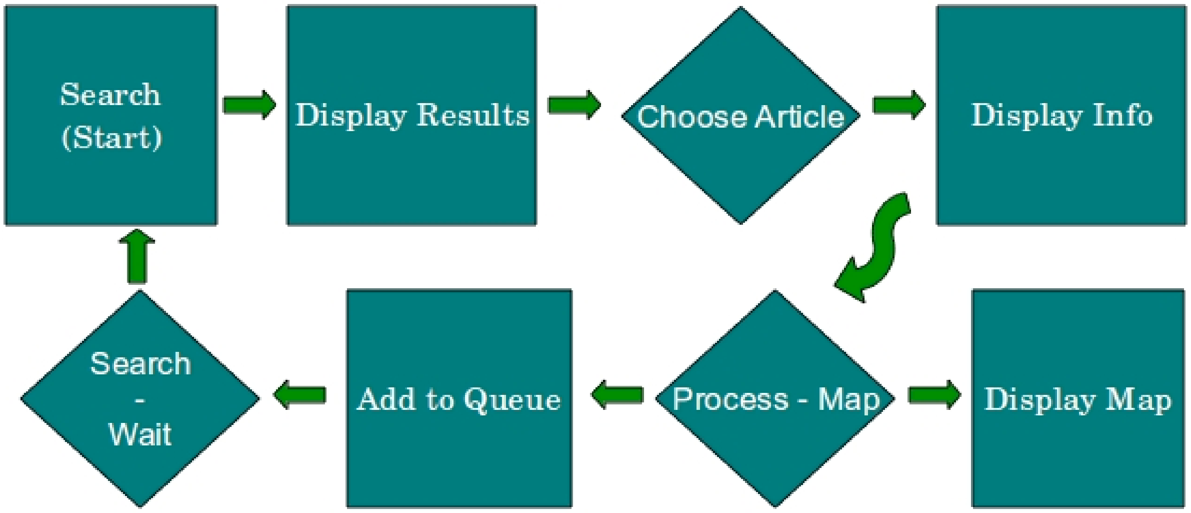
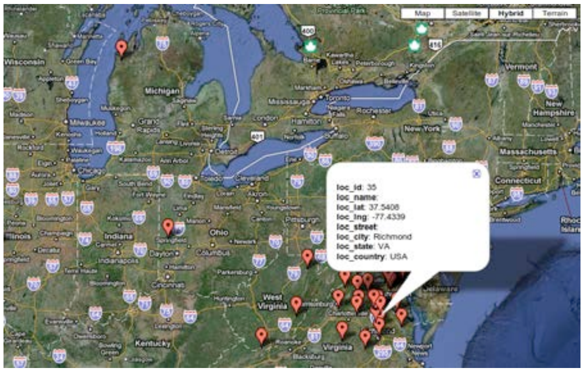


Fig. 5. Website Workflow



loc_id: 35
loc_name:
loc_lat: 37.5408
loc_lng: -77.4339
loc_street:
loc_city: Richmond
loc_state: VA
loc_country: USA

Fig. 3. Fusion Table Map

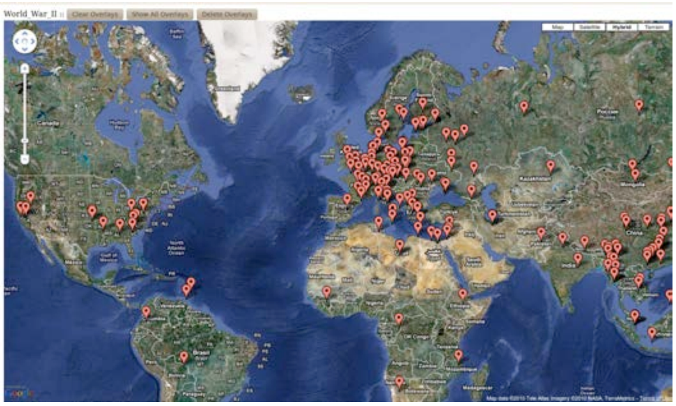


Fig. 4. World War II Map

C. Front End

The current approach we have focuses on maintaining performance and a stable foundation for public use. The frontend is a Ruby on Rails Application which focuses on displaying information to the user. The maps shown on the webpage use geospatial information saved in an offline database. Before the map is displayed, a user can see when the article was last processed, if ever. They may then choose to add an update to the queue, which is managed by a JRuby thread completely independent of the front end. The queue is simply a table in the same database used to store geospatial data that consists of two columns. One is the id of the article that needs to be processed and the other is the time entered into the queue. The table will be consistently queried for the least recent addition to the queue and the JRuby thread will actively update articles. We currently only have one thread processing articles as we experienced problems with multithreading this task. As soon as an article begins to be processed, it is removed from the queue to avoid processing the same article multiple times in parallel when multiple threads are implemented. When idle, we plan to have the thread perform maintenance that will begin with the list of articles in our test set. For each test page, the thread will follow links to other articles found in the page up to a certain depth and process them. The thread will then return to the next test page and do the same. Once these are done, we will probably have it focus on processing articles that have not yet been processed once. The basic workflow for the frontend is shown in Figure 5.

To demonstrate the power of automated extraction and visualization, the World War II article has been recently processed and the map for the article is shown in Figure 4. An infobox is generated for each marker which has the geographical information and the sentence the location was extracted from. The amount of work it would take to hand tag each of these places and add in this sort of information would be extensive. Our program can extract and visualize this information faster we could read a section in the article.

D. Named Entity Recognition

When we choose to “Process” an article as shown in Figure 5, we begin with Witmer’s Geografikos package from his work in [8] for the disambiguation and geocoding of place names. It should be noted that Witmer’s SVM does not identify people and organizations. Thus, a new tool has been integrated into the package for the identification of named entities. We have decided to implement the named entity extraction tool designed by the Stanford Natural Language Processing Group[7]. This package uses a conditional random field (CRF) for named entity recognition (NER) which achieved an f-measure of over 85% for all named entities when tested on the CoNLL 2003 test data [9]. The CRF achieved an f-measure of over 88% in location extraction, substantially higher than the initial phase of Witmer's SVM (67.5%). The disambiguation phase of Witmer's work seems to be very effective but requires some modification to work with this new tool. Taking the approach defined in [8], we will also be implementing a tree structure to weight and disambiguate various possibilities for the correct geospatial entities associated with the extracted names. This process has been improved by:

- Improving the module that cleans wikipedia markup
- Updating the Google Geocode ruby gem to work with Google Maps API v3
- Updating the gem to also return multiple possible locations for a query
- Creating a cache for Google Geocoder to reduce online interaction
- Using a more accurate named entity recognizer which replaced his SVM
- Adding database support for people and organizations
- Incorporated sentence indices into all named entity object models

We would also like to parallelize the process to use multiple threads to optimize speed in the future.

V. Progress and Results

The following list shows our initially proposed tasks for this project:

1) Integrate Stanford NER toolkit into Geografikos package
2) Generate map UI to simply plot all extracted places
3) Write cache for geocoder, parallelize as much as possible
4) Associate named entities with extracted events
5) Develop GUI with temporal sliding bar

Items 1,2, and 3 have been successfully implemented. So far, the Stanford NER toolkit has been performing similarly overall to the LingPipe HMM. The CRF initially encountered errors when processing Gulf War, so these results were excluded from the figures below. In addition, the Fredericksburg article seems to be an abnormality in geocoded results, however it was left in with an f-measure of only 6% because no signs of errors in processing have been found. Raw results are shown in I and post-geocoding results are shown in II. If the Fredericksburg article is dropped from evaluation, post-geocoding the f-measure increases by about 3% when using the Stanford CRF. A comparison between the performance between the CRF and HMM are shown in Figure 6. These articles were used as they were the ones chosen to compare the HMM and SVM in our previous paper. Since the generation of these results, changes have been made to the clean, markup-free text used by the NER and all further stages. The Gulf War article, among other articles that caused problems earlier can now be processed successfully. Some previous errors were due to errors in expanding Wikipedia infoboxes which allowed us to generate an extensive list of related Wikipedia articles. For now, we have removed infobox expansion and simply delete infoboxes from articles and do no further processing on them. Changes like these have drastically changed the markup-free versions of the articles. Our method in evaluating the performance of the NER alone and the geocoding process was heavily dependent on the generated cleaned text matching the hand tagged data. Thus, statistics are not being generated at this point but based on the human examination of results, we can see it has not reduced performance.

TABLE I
Raw Geospatial NE Results

	Precision	Recall	F-Measure
HMM Results	0.489	0.615	0.523
CRF Results	0.579	0.575	0.572

TABLE II
Resolved Geospatial NE Results

	Precision	Recall	F-Measure
HMM Results	0.878	0.734	0.796
CRF Results	0.954	0.695	0.791

It should be noted that the new CRF results make use of the cache which should not effect these statistics but decreased the number of online geocoding queries by 70%. Since the majority of locations can be found in the cache, we have also eliminated a great number of delays that are normally required to avoid running into Google Geocoder's throughput limits. At the moment, this delay is set to 0.1 seconds so we end up saving a few seconds per article. More detailed performance evaluations still need to be made to more clearly identify how well the cache has improved efficiency. Initially, once a place was geocoded and resolved once, it was assumed that all future references to the name of that location were referencing the same geospatial entity. This means that if there are two different places mentioned with the same name, both will appear as the first one geocoded. We have since fixed this problem by saving the list of results returned by the geocoder, rather than the specific location identified after disambiguation.

Named entities have been associated with events through their location in a sentence. If for example, a location and event are both identified in the third sentence of an article, they are assumed to be related. When visualizing this information, one event can have many locations associated with it and one location can also relate to various events. One of the problems with our process is that many specific location instances are being missed. Even if a location is identified in one sentence, it may not have been tagged a location in another sentence that actually has an event is in. For example if the city of Denver is tagged as a location in sentence 3 and saved to the database, future examples of Denver are not automatically tagged because of its initial identification. Thus, if an actual event happens in Denver and is correctly tagged, no

[7] urlhttp://nlp.stanford.edu/

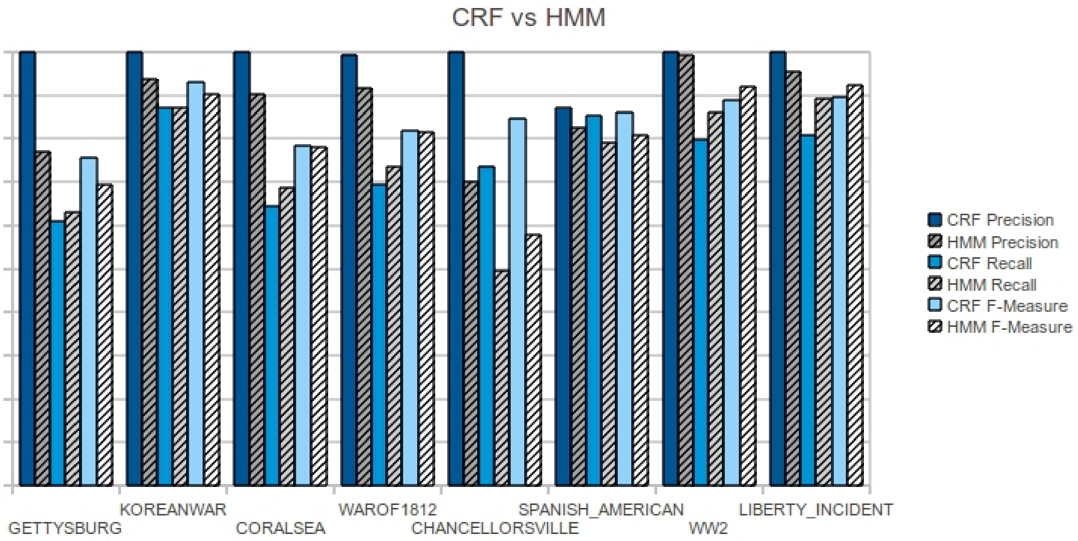


Fig. 6. Basic Format of GUI

re-processing of the sentence is done to confirm that no other NE's exist in it. We plan on fixing this issue in the future.

VI. Future Experiments

To measure the performance of the tool we need to separate it into the different parts. Currently, we use the same measures for precision and recall of the NER as Witmer. However, this only measures the precision and recall for locations. We need to measure the performance of the NER for people and organizations in this subject area as well. To do this we will need to obtain various hand tagged historical Wikipedia articles, a handful of which have already been tagged for places. We need to add in tags for people and organizations to about twenty hand tagged articles. If these tests go well, we can also hand tag news articles in a similar fashion. To assess the event extraction, we will need to hand tag important events within the same document as the hand tagged NE's. We can then check to see if the phrase we pulled out to describe the event included the NE's, and whether the NER had even identified them. Much of processing can be done before public release of the tool as we are storing the information after processing. Thus, we plan to process an initial set, if not all the English Wikipedia articles we currently have access to. This will probably be on the scale of hundreds of thousands of articles, if not millions. Thus, we need to keep track of the number of articles processed over time to measure the throughput with these added features. We can also do this to archived news articles. In addition, we will need multiple testers to utilize the GUI and evaluate the helpfulness and usability of the front end. We can measure these attributes with a simple rating system, perhaps a 1 through 5 rating.

VII. Future Work

We have identified a number of ways to improve the different aspects of our system:

1) Incorporate temporal refinement of events and locations displayed in the GUI
2) Evaluate the NER for all types of NE's in our test articles
3) Display a picture of the people associated with an event
4) Retrieve and display information on these people (D.O.B., D.O.D., etc.)
5) Determine a "page focus" and use it to center the map, etc.
6) Rework the disambiguation of places
7) Implement sub-sentence level event detection
8) Add feature for retrieving article text straight from Wikipedia

Our plans for future work is still similar to the one initially proposed. However, a much more specific implementation has been planned along with many improvements we did not originally foresee. We would like users to be able to search articles and add them to a list. At any point, the user should be able to choose to map all the locations. On the side of the page, a list will be displayed with checkmarks next to each article. At any time, the status of the checkmark will represent the presence of that article's locations on the map. For example if I have the Battle of Fredericksburg article selected, all the locations from the article should appear on the map. If I uncheck it, those locations should disappear unless of course, a location is also referenced in another selected article that I have checked. Another tool may offer a user the option to select a region on a map and a list of articles that reference places within that area will be listed, if not spefic events that occurred there.

Then of course there is the aspect of temporal information as well. In regard to item 1, we wish to allow the refinement of geospatial searches through temporal parameters. For example, instead of displaying all events that occur in Denver, Colorado, we can limit to showing events in the last ten years. This will also allow us to eventually create a playback feature which will allow one to see where events were occuring over time in the context of articles.

In regard to items 3 and 4, retrieving this type of information will be a complicated task. Many problems arise in biographical information retrieval because the ambiguity of names. Some historical figures may be returned at the top of a list of results but the most likely figure may not always be the one being referenced. We will look into disambiguating people further in the future. For now, our idea for an approach in Wikipedia articles is following Wikipedia links. A list of links is extracted when the text of a page is cleaned of markup. If a person is identified and a link matching that name is also extracted, we can follow that link. If the person's page is confirmed to be about that person, we may be able to extract information from Wikipedia infoboxes which follow a general template dependent on the topic of the article.

Item 5 is very important in light of improving our GUI. Amitay et al. discuss a method for determining a focus area for a page by using first disambiguating place names [10]. They then each "geographic mention" is "disambiguated into a taxonomy node" where places are scored for importance. Ultimately, their method can make the distinction between article that focuses on a very specific place or a larger region. Their method can also retrieve multiple foci. Finding a page focus can be used to center the map when it is initially created, along with helping to categorize articles together that focus on the same region. In the future, we would like to be able to select a region on the map and see what articles reference that region. Rather than having a broad list of articles, many of which that only briefly mention a place and do not actually focus on it, it would be beneficial to have a refined list of articles in which that region is of significant importance.

The disambiguation of places refernced in item 7 is the tree structure implemented by Witmer. Although the idea was implemented at some point, we do not have access to this portion of his code. Thus, it will be reworked and the Geografikos package will be modified to handle multiple locations returned by Google Geocoder, rather than only one.

Currently, only sentence level event detection has been implemented. This means that if an event is detected within a sentence, the event's location in the article is labeled by sentence and only one event can occur per sentence. This also means that all named entities within that sentence are also associated with that event. Thus, if one event constitutes one half of the sentence and references a specific location at a certain time while another event constitutes the other half at another place at another time, everything is rolled into one event that has both places associated with it but only one time is chosen. Although much of the information will still be gathered and displayed, we do lose some in the process. Thus, we would like to improve our event detection and relation to named entities so that it can handle sub-sentence information.

At the moment, the document text displayed when you search for a Wikipedia article is the clean text generated from our saved version of the article. This means that more recent articles and updates are not currently in our database. In addition, all the regular pictures, links, etc. displayed in an article being viewed on the Wikipedia website are absent from ours. In the future, we would like to port this application into a plug-in or a frame that allows one to navigate Wikipedia with regular convenience but have access to our database of information and visualization tools. The search featuers of Wikipedia and website overall are far more functional than ours. Rather than implementing methods for duplicating their features and actively updating our articles from their database, we would like to make our application more independent. This will also pave the way for portability to other online sources of information, such as news articles. Modifying our application to dynamically retrieve information from a browser and process it would prove to be a much greater benefit to users. The current problems associated with this is that our object models are generated from data stored in a database. A list of events and named entities is stored in the database so that they are easily accessibly from a page object. Handling multiple sources of online information that is constantly changing will add a great deal of processing that our system is not currently set up to manage. In addition, each website will have different types of markup. Our current system will only clean markup that is commonly found in Wikipedia articles. As sources of information become less structured, processing them and taking advantage of existing features becomes a great deal harder. Infoboxes, Wikipedia links to other articles, and peer review are some featuers that we currently expect to be there. In the future, we will have to take advantage of similar features or handle their absence to help reduce the difference in quality between processing one source or another.

VIII. Discussion

This project has opened up a great many possibilities for future work while still offering a utility that can be publically accessible with few modifications. Although many improvements will require time to implement, we have a very good idea of where to take this tool. A very important factor seems to be the quality of the named entity recognizer used in the most initial stage of processing. By finalizing the module that cleans markup from Wikipedia articles, we can justify the dedication of time to hand tagging events and all named entities in our training corpus and future corpora. We can then evaluate the NER more accurately and begin experiments in improving it by retraining its model. In addition, we have found a few problems with Google Geocoder. Currently, a region bias must be specified for geocoding results. The bias currently defaults to the United States. This means that we cannot geocode "Cambridge" and have the geocoder return "Cambridge, UK" without explicitely changing the bias or including "UK" in the query. Thus, we may need to change geocoders or find a way around this if possible. In addition, the google-geocode ruby gem has not been updated in the online repostiroy to be compatible with the new Google Maps API Javascript v3. We have already implemented this and need to submit it for inclusion in the online gem repository. Although the temporal refinement of locations and events through the sliding bar shown in the proposed GUI has not been implemented yet, we are very close to enabling this feature. At this point, all of our initial goals will be met and we can focus on new ones.

IX. Conclusion

Ultimately we have begun creation of a tool that allows the extraction and visualization of important facts in an article. Our corpus currently only consists Wikipedia articles but will eventually be expanded to other encyclopedias and even other forms of text. The goal is to enable users to gather a quick overview of what places, people, and time periods are involved in an article, whether to substitute or supplement the main text. The project has required a level of cooperation to combine work on event extraction and named entity extraction, but we have successfully combined the information. We are nearing a decently functional GUI that accurately and efficiently displays extracted events and associated entities. The process of actually extracting these events is currently slower than planned, however extracting named entities is still running in the terms of seconds. Most articles process between 5-20 seconds. Progress did slow down as the depth of the work increased but many improvements and new ideas have surfaced in the process. The GUI is not quite as polished and functional as we originally hoped, but it should be completed a short time after this summer. We have successfully associated events and named entities and done basic visualization of these. The work in the temporal refinement is currently the focus of the work. There are many things we can do in the future, some which will take minimal amounts of time and some that will take longer. Taking into account the rate of progression this summer, we should be able to implement most ideas listed in future work during this upcoming fall semester. This of course, is with only one person working on the project part time. The website should be publically available and work relatively well at the end of the summer and fully developed in winter.

References

[1] S. Auer and J. Lehmann, "What have innsbruck and leipzig in common? extracting semantics from wiki content," in *ESWC '07: Proceedings of the 4th European conference on The Semantic Web*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 503–517.

[2] R. Mihalcea and A. Csomai, "Wikify!: linking documents to encyclopedic knowledge," in *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. New York, NY, USA: ACM, 2007, pp. 233–242.

[3] *The Geospatial Web: How Geobrowsers, Social Software and the Web 2.0 are Shaping the Network Society (Advanced Information and Knowledge Processing)*. Springer, 2007. [Online]. Available: http://www.amazon.com/Geospatial-Web-Geobrowsers-Information-Processing/dp/1846288266%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1846288266

[4] Y.-F. R. Chen, G. Di Fabbrizio, D. Gibbon, S. Jora, B. Renger, and B. Wei, "Geotracker: geospatial and temporal rss navigation," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 41–50.

[5] W. Dakka and S. Cucerzan, "Augmenting wikipedia with named entity tags," *IJCNLP*, 2008.

[6] D. Klein, J. Smarr, H. Nguyen, and C. D. Manning, "Named entity recognition with character-level models," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*. Morristown, NJ, USA: Association for Computational Linguistics, 2003, pp. 180–183.

[7] G. Zhou and J. Su, "Named entity recognition using an HMM-based chunk tagger," in *Proc. 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, 2002.

[8] J. Witmer and J. Kalita, "Extracting geospatial entities from wikipedia," *IEEE International Conference on Semantic Computing*, pp. 450–457, 2009.

[9] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Morristown, NJ, USA: Association for Computational Linguistics, 2005, pp. 363–370.

[10] E. Amitay, N. Har'El, R. Sivan, and A. Soffer, "Web-a-where: geotagging web content," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 2004, pp. 273–280.