

Proceedings of the Seminar

Machine Learning: Theory and Applications

University of Colorado, Colorado Springs

August 5, 2016

Editor: Jugal K. Kalita

Funded by

National Science Foundation

Preface

It is with great pleasure, we present to you papers describing the research performed by the NSF-funded Research Experience for Undergraduates (REU) students who spent 10 weeks during the summer of 2016 at the University of Colorado, Colorado Springs. Within a very short period of time, the students were able to choose cutting-edge projects involving machine learning, write proposals, design interesting algorithms and approaches, develop code, and write papers describing their work. We hope that the students will continue working on these projects and submit papers to conferences and journals within the next few months. We also hope that it is the beginning of a fruitful career in research and innovation for all our participants.

This year, we were also fortunate to have received supplemental funding to invite three teachers from Academy School District 20 in Colorado Springs to join the summer program for six weeks, providing them with precious time to perform research, mingle and discuss with other researchers, and most importantly, ponder over ideas regarding how topics in machine learning or learning from data, can be introduced to K-12 students.

We thank the National Science Foundation for funding our REU project. We also thank the University of Colorado, Colorado Springs, for providing an intellectually stimulating environment for research. In particular, we thank Drs. Terrance Boulton, Rory Lewis, Kristen Walcott-Justice, and Qing Yi who were faculty advisors for the REU students. We also thank Alessandra Langfels for working out all the financial details. We also thank our graduate students, in particular, Feras AlTarouti, Tri Doan and Ethan Rudd, for helping the students with ideas as well as systems and programming issues. Francisco Torres-Reyes and his team also deserve our sincere gratitude for making sure that the computing systems performed reliably during the summer.

Sincerely,

Jugal Kalita
jkalita@uccs.edu
Professor
August 5, 2016



NSF REU-RET Seminar on Machine Learning

Department of Computer Science
University of Colorado, Colorado Springs
Osborne Center, UCCSTeach Room



August 5, 2016: Friday

10:45-10:50 AM: Welcome Remarks

10:50-12:10 AM Session Chair: *Lisa Jesse, Co-founder, Intelligent Software Solutions, Inc., Colorado Springs, CO*

10:50-11:10 *David Foley, Kutztown University of Pennsylvania, Kutztown, PA: Integrating WordNet for Multi-sense Word Embeddings in Vector Semantics*

11:10-11:30 *Kyra Yee, Pomona College, Claremont, CA: Composition of Compound Nouns Using Distributional Semantics*

11:30-11:50 *Nathan Harmon, University of Colorado, Colorado Springs, CO: Computation Pattern Classification for Optimization*

11:50-12:10 *Matthew Simpson, North Carolina State University, Raleigh, NC: Application of Machine Learning to Choice of Algorithms in Computational Software*

12:10-1:00 PM: Lunch: Chancellor Dr. Pamela Shockley-Zalaback of the University of Colorado, Colorado Springs, will join us briefly during lunch.

1:00-2:20 PM Session Chair: *Dr. Kristen Walcott-Justice, Assistant Professor, University of Colorado, Colorado Springs, CO*

1:00-1:20 *Kamal Kamalaldin, Kalamazoo College, Kalamazoo, MI: Classifying and Localizing Epileptic Brain States Using Structural Features of Neuronal Sugihara Causation Networks*

1:20-1:40 *Alayna Kennedy, Pennsylvania State University, State College, PA: Optimization of Neural Network Architecture for Biomechanics Classification Tasks with Electromyograph Inputs*

1:40-2:00 *Liam Schramm, Bard College, Annandale-on-Hudson, NY: Improving Performance of Search Repair Using Fast Learned Heuristics*

2:00-2:20 *Cora Coleman, New College of Florida, Sarasota, FL: Improving Coverage of Fuzz Testing to Determine Security Vulnerabilities in Mobile Applications*

2:20-2:30 PM: Break

2:30-3:40 PM Session Chair: *Dr. Rory Lewis, Associate Professor, Computer Science, University of Colorado, Colorado Springs, CO*

2:30-2:50 *Andrea Costas, Rice University, Houston, TX: Improving Partial Fingerprint Recognition*

2:50-3:10 *Abigail Graese, University of Colorado, Colorado Springs, CO: Assessing Threat of Adversarial Examples on Deep Neural Networks*

3:10-3:40 *Simon Ellis, Andrea McGeorge and Celeste Puccio, Academy International Elementary School, Colorado Springs, CO: A Proposal for Implementing Machine Learning Educational Application in an Elementary School Setting*

3:40 PM: Closing Remarks

7:00 PM: Farewell Dinner, Restaurant to be announced

Table of Contents

<i>Integrating WordNet for Multi-sense Word Embeddings in Vector Semantics</i> David Foley and Jugal Kalita.....	1
<i>Composition of Compound Nouns Using Distributional Semantics</i> Kyra Yee and Jugal Kalita.....	6
<i>Computation Pattern Classification for Optimization</i> Nathan Harmon, Qing Yi and Jugal Kalita.....	13
<i>Automatic Algorithm Selection in Computational Software Using Machine Learning</i> Matthew Simpson, Qing Yi and Jugal Kalita.....	16
<i>Classifying and Localizing Epileptic Brain States Using Structural Features of Neuronal Sugihara Causation Networks</i> Kamal Kamalaldin, Rory Lewis, Chad Mello, Dorottya Cserpan, Somogyvari Zoltan, Peter Erdi and Zsolt Borhegyi.....	26
<i>Optimization of Neural Network Architecture for Biomechanics Classification Tasks with Electromyograph Inputs</i> Alayna Kennedy and Rory Lewis.....	31
<i>Improving Performance of Search Repair Using Fast Learned Heuristics</i> Liam Schramm and Jugal Kalita.....	38
<i>An Open Set Protocol for Improving Malware Classification in Intrusion Detection</i> Cora Coleman, Ethan Rudd and Terrance Boulton.....	44
<i>Improving Partial Fingerprint Recognition</i> Andrea Costas and Terrance Boulton.....	51
<i>Assessing Threat of Adversarial Examples on Deep Neural Networks</i> Abigail Graese, Andras Rozsa and Terrance Boulton.....	57
<i>A Proposal for Implementing Machine Learning Educational Application in an Elementary School Setting</i> Simon Ellis, Andrea McGeorge and Celeste Puccio.....	63

Integrating WordNet for Multiple Sense Embeddings in Vector Semantics

David Foley*, Jugal Kalita†

*Kutztown University

†University of Colorado Colorado Springs

Abstract—Popular distributional approaches to semantics allow for only a single embedding of any particular word. A single embedding per word conflates the distinct meanings of the word and their appropriate contexts, irrespective of whether those usages are related of completely disjoint. We compare models that use the graph structure of the knowledge base WordNet as a post-processing step to improve vector-space models with multiple sense embeddings for each word, and explore the application to word sense disambiguation.

Index Terms—Computational Linguistics, Vector Semantics, WordNet, Synonym Selection, Word Sense Disambiguation

I. INTRODUCTION

Vector semantics is a computational model of written language that encodes the usage of words in a vector space, which facilitates performing mathematical manipulations on words as vectors [7]. These vectors encode the contexts of words across a corpus, and are learned based on word distributions throughout the text. Vectors can then be compared by various distance metrics, usually the cosine function, to determine the similarity of the underlying words. They also seem to possess some modest degree of compositionality, in the sense that the addition and subtraction of vectors can sometimes result in equations that appear to reflect semantically meaningful relationships between words [11], [12]. Because it allows for the use of these well studied techniques from linear algebra to be brought to bear on the difficult domain of semantics, vector space models (VSMs) have been the focus of much recent research in NLP.

While vector representations of word meaning are capable of capturing important semantic features of words and performing tasks like meaning comparison and analogizing, one of their shortcomings is their implicit assumption that a single written word type has exactly one meaning (or distribution) in a language. But many words clearly have different senses corresponding to distinct appropriate contexts. Building distributional vector space models that account for this polysemous behavior would allow for better performance on tasks involving context-sensitive words, most obviously word sense disambiguation. Previous research that attempted to resolve this issue is discussed at length in the next section. Most common methods either use clustering or introduce knowledge from an ontology. The goal of the present research is to develop or improve upon methods that take advantage of the semantic groups and relations codified in WordNet, and specifically to focus on the downstream WSD task, which

is often neglected in favor of less useful similarity judgment evaluations.

The algorithm we examine in depth can in principle be implemented with any ontology, but in the present paper we focus exclusively on WordNet. WordNet (WN) is a knowledge base for English language semantics [15]. It consists of small collections of synonymous words called synsets, interconnected with labeled links corresponding to different forms of semantic or lexical relations. We will be particularly interested in the synset relation of hypernymy/hyponymy. Hyponyms can be thought of as semantic subsets: If A is a hyponym of B, then x is A implies x is B, but the converse is not true. WordNet is also equipped with a dictionary definition for each synset, along with example sentences featuring varying synonymous words. Often implementations that use WordNet’s graph structure fail to make use of these other features, which we will show can improve performance on several tasks.

II. RELATED WORK

Our work is based primarily on that of Jauhar et al’s RETROFIT algorithm [6], which is discussed at greater length in Section 3. Below we discuss previous models for building sense embeddings.

A. Clustering-Based Methods

Reisinger and Mooney [16] learn a fixed number of sense vectors per word by clustering context vectors corresponding to individual occurrences of a word in a large corpus, then calculating the cluster centroids. These centroids are the sense vectors.

Huang et al. [5] build a similar model using k-means clustering, but also incorporate global textual features into initial context vectors. They compile the Stanford Contextual Word Similarity dataset (SCWS), which consists of over two thousand word pairs in their sentential context, along with a similarity score based on human judgments from zero to ten.

Neelakantan et al. [13] introduce an unsupervised modification of the skip-gram model [9] to calculate multiple sense embeddings online, by maintaining clusters of context vectors and forming new word sense vectors when a context under consideration is sufficiently far from any of the word’s known clusters. The advantage of the method is that it is capable of detecting different numbers of senses for different words, unlike the previous implementations of Huang et al. and Reisinger and Mooney.

B. Ontology-Based Methods

Chen et al. [3] first learn general word embeddings from the skip-gram model, then initialize sense embeddings based on the synsets and glosses of WN. These embeddings are then used to identify relevant occurrences of each sense in a training corpus using simple-to-complex words-sense disambiguation (S2C WSD). The skip-gram model is then trained directly on the disambiguated corpus.

Rothe and Shutze [17] build a neural-network post-processing system called AutoExtend that takes word embeddings and learns embeddings for synsets and lexemes. Their model is an autoencoder neural net with lexeme and synset embeddings as hidden layers, based on the intuition that a word is the sum of its lexemes and synset is the sum of its lexemes.

Our intuitions are most similar to those of Jauhar et al [6] and we will be building on one of their approaches. Their RETROFIT algorithm learns embeddings for different word senses from WN by iteratively combining general embeddings according to the graph structure of WN. The approach is discussed in more detail below.

III. IMPLEMENTATION

A. RETROFIT

Because our work follows so directly from [6], we repeat the essential details of the RETROFIT algorithm here. Let $\Omega = (S_\Omega, E_\Omega)$ be an undirected graph. We call Ω an *ontology* when the set of vertices S_Ω represent semantic objects of some kind and the set of edges E_Ω represent relationships between those objects. In the case of WN, S_Ω is the set of synsets and E_Ω are the semantic links (notably hypernyms and hyponyms). Given a set of sense-agnostic word embeddings \hat{V} and an ontology Ω , RETROFIT infers a set of sense embeddings \hat{S} that is maximally "consistent" with both \hat{V} and Ω . By "consistency" we refer to the minimization of the objective function

$$D(\hat{S}) = \sum_{ij} \alpha \|\hat{w}_i - \vec{s}_{ij}\|^2 + \sum_{ij} \sum_{i'j' \in N_{ij}} \beta_r \|\vec{s}_{ij} - \vec{s}_{i'j'}\|^2$$

where N_{ij} is the set of neighbors of s_{ij} defined in E_Ω and α and β are hyperparameters controlling the importance of initial sense-agnostic embeddings and various ontological relationships, respectively. Essentially RETROFIT aims to make a sense embedding as similar to its sense-agnostic embedding as possible, while also reducing the distance between related senses as defined by Ω . It achieves this by iteratively updating sense embeddings according to

$$\vec{s}_{ij} = \frac{\alpha \hat{w}_i + \sum_{i'j' \in N_{ij}} \beta_r \vec{s}_{i'j'}}{\alpha + \sum_{i'j' \in N_{ij}} \beta_r} \quad (2)$$

until convergence. The RETROFIT implementation discussed in [6] defines only synonym, hypernym and hyponym relations, with respective weights of $\beta_r = 1.0, 0.5$ and 0.5 . Below we discuss several of the limitations associated with this RETROFIT implementation and possible improvements.

1) *Impoverished Synsets*: Many word senses are relatively isolated in the WordNet structure. They occur in synsets with few or no synonyms or semantic relations. In the case that the word has only one meaning, this isn't a problem, because the sense-agnostic embedding is in that case unambiguous. But in the case that the word has one or more other semantically rich senses (ie, senses with synonyms and hyper/hyponym relations), the impoverished sense is unduly influenced by the general embedding and its unique meaning is not distinguishable. In the extreme case both senses are identical.

2) *Compound Words and Multi-word Lemmas*: The original RETROFIT implementation discards multi-word lemmas (and entire synsets if they consist only of multi-word lemmas.) But there exist synsets for whom most or all of the related WN synsets contain only multi-word lemmas. (E.g. In the case of *brass.n.01*, the hyponyms are almost all compound words for types of brass.) Adjusting the RETROFIT algorithm to allow for embeddings of the multi-word lemmas that appear in WN would greatly reduce the number of impoverished synsets.

3) *Underrepresented Senses*: The general embedding produced by word2vec conflates all usages of a word. If a particular sense of a word is significantly less common than others, the word2vec embedding will not be a good representation of the sense. RETROFIT indiscriminately tries to minimize the distance from any particular sense and its word2vec embedding.

For these reasons we make the following modifications to RETROFIT:

1) Regardless of the position of a word sense in WordNet, it will be equipped with a descriptive gloss that clarifies its usage. We incorporate content words from each synset's gloss in the RETROFIT algorithm's objective function.

2) We implement a naive model to handle a compound word by simply representing its sense-agnostic embedding as the average of the sense-agnostic embeddings of its constituent words. Although this is obviously inadequate for many compound words, we find it is already an improvement.

3) The sense-agnostic embedding of a word is assumed to be the weighted average of its sense embeddings, proportional to how common a particular word sense is. We calculate the sense-frequencies from the SemCor corpus, which consists of around 300,000 words tagged with their WordNet 3.0 synsets [10].

B. Weighted RETROFIT

Let $M = (V, \hat{V}, S, \hat{S}, P, \Omega)$ be a model consisting of a vocabulary V and sense-agnostic embeddings \hat{V} , a set of word senses S and sense-embeddings \hat{S} , a discrete probability density function $P : V \times S \rightarrow \mathbb{R}$, and an ontology Ω . We seek the set \hat{S} that minimizes the new objective function for the weighted RETROFIT algorithm

$$\begin{aligned}
 D(M) = & \sum_i \alpha \left\| \hat{w}_i - \sum_j p_{ij} \vec{s}_{ij} \right\|^2 \\
 & + \sum_{ij} \sum_{i'j' \in N_{ij}} \beta_r \left\| \vec{s}_{ij} - \vec{s}_{i'j'} \right\|^2 \\
 & + \sum_{ij} \sum_{i' \in G_{ij}} \gamma \left\| \hat{w}_{i'} - \vec{s}_{ij} \right\|^2
 \end{aligned} \quad (3)$$

by iteratively updating embeddings according to

$$\vec{s}_{ij} = \frac{\alpha p_{ij} \hat{w}_i - \alpha p_{ij} \sum_{k \neq j} p_{ik} \vec{s}_{ik} + \sum_{i'j' \in N_{ij}} \beta_r \vec{s}_{i'j'} + \gamma \sum_{i' \in G_{ij}} \hat{w}_{i'}}{\alpha p_{ij}^2 + \sum_{i'j' \in N_{ij}} \beta_r + \sum_{i' \in G_{ij}} \gamma} \quad (4)$$

where $\hat{w}_i \in \hat{V}$, $\vec{s}_{ij} \in \hat{S}$, $p_{ij} = P(s_{ij}|w_i)$, N_{ij} is the set of neighbor indices of the j th sense of the i th word defined in Ω , $G_{ij} = \{i : w_i \in \hat{V} \text{ is in the gloss of } s_{ij}\}$ and α , β_r and γ are the parameters controlling the weights of sense-agnostic word embeddings, relations and gloss words respectively. Note that iteratively updating the sense embeddings via Eqs. 2 or 4 is equivalent to optimizing their respective objective functions via coordinate descent.

IV. EVALUATION

We train three variations of the RETROFIT algorithm on the 50-dimensional global context vectors produced by Huang et al [5]: the unmodified RETROFIT, RETROFIT with gloss words and multi-word lemmas, and RETROFIT with weighted senses as discussed above. Training time is similar between the first two; weighted RETROFIT takes about twice as long. All converge to a solution within 0.01 within fifteen iterations.

The models are evaluated on two different tasks: Synonym Selection and Word Sense Disambiguation. We first include and discuss results from some similarity judgment tasks, but these serve more as stepping stone than as a rigorous measure of model quality. Faruqui et al. [4] give a comprehensive assessment of the inadequacies of evaluating the quality of embeddings on word similarity tasks. In general, these tasks are fairly subjective and a model's performance on them does not correlate with performance on downstream NLP tasks.

A. Similarity Judgments

We evaluate the models on two word-similarity tasks: RG-65 and SCWS. The RG-65 dataset [18] consists of sixty-five pairs of words and an average human judgment of similarity scaled from one to four. The Stanford Contextual Word Similarity dataset [5] consists of over two thousand word pairs in their respective sentential contexts and an average human evaluation of similarity from one to ten.

Evaluation on the RG-65 dataset is a straightforward calculation of the average cosine similarity of each pair of sense embeddings, as used by Jauhar et al. [6] and originally proposed by Reisinger and Mooney [17]. As an exploration,

	Similarity Judgments		SCWS	
	RG-65		CXT	AVG
	AVG	MAX		
RETROFIT	0.73	0.79	0.50	0.58
gloss + multi RETROFIT	0.72	0.85	??	??
Weighted RETROFIT	0.69	0.84	??	??

TABLE I

PERFORMANCE ON RG-65 AND SCWS WORD SIMILARITY DATASETS. SCORES ARE SPEARMAN'S RANK CORRELATION.

	Synonym Selection	
	ESL-50	TOEFL
RETROFIT	64.0	68.75
gloss + multi RETROFIT	62.0	81.25
Weighted RETROFIT	60.0	75.0

TABLE II

PERCENT ACCURACY ON ESL-50 AND TOEFL SYNONYM SELECTION USING MAXSIM COMPARISON

we also consider the results of using the maximum cosine similarity.

We evaluate performance on SCWS by first disambiguating the words in their contexts, then comparing the cosine similarity of the chosen sense vectors. Words are disambiguated using the simple-to-complex algorithm (S2C) described by Chen et al. in [3]. S2C disambiguates every word in a sentence in increasing order of ambiguity, where a word is considered more ambiguous if it has more senses defined in WN. First, a context vector is initialized by averaging the general embeddings of each word, then the least ambiguous word is assigned a sense from WN based on which sense embedding has the greatest cosine similarity with the context embedding. The context embedding is then updated as the average of the general embeddings of the ambiguous words and the sense embeddings of the disambiguated words. The only parameter of the model is a confidence threshold. At each disambiguation, if the difference between the rating of two candidate senses for a word are within the confidence threshold of each other, we choose not to disambiguate the word and continue to use its general embedding as the context in subsequent iterations.

Our results are displayed in Table 1.

B. Synonym Selection

We test the models on two synonym selection datasets: ESL-50 [19] and TOEFL [8]. ESL-50 is a set of fifty English sentences with a target word for which a synonym must be selected from four candidate words. TOEFL consists of eighty context-independent words and four potential candidates for each. For both datasets, we use the same maxSim selection criteria as Jauhar et al [6]. We select the sense vector \vec{s}_{ij} that corresponds to:

$$\text{maxSim}(w_i, w_{i'}) = \max_{j,j'} \cos(\vec{s}_{ij}, \vec{s}_{i'j'})$$

Our results are presented in Table 2.

C. Word Sense Disambiguation

We use Semeval 2015 task 13 [9] as our English WSD test. The corpus for the task consists of four documents taken

	Word Sense Disambiguation				
	Nouns	Verbs	Adjectives	Adverbs	All
MFS	45.8	49.9	67.5	70.6	53.5
RETROFIT	49.1	57.0	67.3	75.3	56.2
Modified RETROFIT	50.6	50.0	69.2	76.5	57.0
Weighted RETROFIT	50.0	52.8	65.4	76.5	56.8

TABLE III

SEMEVAL 2015 TASK 13 F1 SCORES OF THE MODELS USING THE CONTEXTMAX DISAMBIGUATION FUNCTION.

	Word Sense Disambiguation				
	Nouns	Verbs	Adjectives	Adverbs	All
RETROFIT	52.5	57.2	77.3	77.8	61.1
Modified RETROFIT	53.6	56.4	76.0	79.0	61.6
Weighted RETROFIT	53.9	59.2	75.4	77.8	62.1

TABLE IV

SEMEVAL 2015 TASK 13 F1 SCORES OF THE MODELS USING THE CONTEXTMAX DISAMBIGUATION FUNCTION, RESTRICTED TO CORRECT POS

	Nouns	Verbs	Adjectives	Adverbs	All
RETROFIT	49.5	49.2	64.2	79.0	55.7
Modified RETROFIT	54.8	50.0	67.9	77.8	59.5
Weighted RETROFIT	53.0	52.4	62.3	74.1	57.9

TABLE V

SEMEVAL 2015 TASK 13 F1 SCORES OF THE MODELS USING THE LOCALGLOBAL DISAMBIGUATION FUNCTION

	Nouns	Verbs	Adjectives	Adverbs	All
RETROFIT	52.2	55.6	73.5	80.2	60.2
Modified RETROFIT	56.6	57.6	74.1	80.2	63.4
Weighted RETROFIT	55.6	59.2	72.9	76.5	62.1

TABLE VI

SEMEVAL 2015 TASK 13 F1 SCORES OF THE MODELS USING THE LOCALGLOBAL DISAMBIGUATION FUNCTION, RESTRICTED TO CORRECT POS

from the biomedical, mathematical and social issues domains, annotated with part of speech information. The task also includes named entity recognition, which we do not handle, except in the incidental case where there is a WN synset for a named entity. We explore two different methods for WSD. The first chooses a word sense by identifying a word that co-occurs in the sentence and has a sense that is closest to a sense of our target word. The intuition of the model is that although particular words may be totally unrelated to the sense of the target word, there should exist somewhere in the sentence a word pertaining to the subject described by the ambiguous word. Formally, this method is described as the *contextMax* function:

$$contextMax(w, c) = \arg \max_{s \in S_i} (\max_{\substack{c \in \bigcup_{k \neq i} S_k}} \cos(\vec{s}, \vec{c}) \cdot p(s|w))$$

where S_i is the set of senses of the i th word of the context sentence.

The second WSD method incorporates both local and global context in equal parts. The intuition is that nearby words in a particular sentence will capture information about the particular usage of a word, while words that appear over the course of a passage will characterize the subject matter being discussed. Both of these components are essential to human understanding and should aid WSD algorithms, as discussed in [20]. Formally, we define the localGlobal WSD function as

$$localGlobal(w, c) = \arg \max_{s \in W_{ij}} (\cos(\vec{s}, \vec{c}_{ij}) \cdot p(s|w))$$

where the context vector \vec{c}_{ij} for the j th word of the i th sentence is given by

$$\vec{c}_{ij} = \frac{\vec{l}_{ij}}{|\vec{l}_{ij}|} + \frac{\vec{g}_i}{|\vec{g}_i|}$$

and

$$\vec{l}_{ij} = \sum_{k \neq j} \frac{1}{|j - k|} \hat{w}_{ik}$$

$$\vec{g}_i = \sum_{n=i-2}^{i+2} \sum_k \hat{w}_{nk}$$

As a baseline we compare against the most-frequent sense tagger (MFS) trained on the Semcor corpus [9], defined simply as

$$mfs(w) = \arg \max_{s \in S_w} (p(s|w))$$

Tables 3 and 4 display results for our models when unrestricted. Tables 5 and 6 show results when the search is restricted by part of speech information. Results are ranked by F1 score, the harmonic mean of precision and recall.

By all measures, the various RETROFIT implementations outperform the MFS baseline. Weighted RETROFIT and Modified RETROFIT both improve the initial model. The best performing systems on the Semeval 2015 task 13 English corpus are LIMSI and SUDOKU [9], which achieve F1 scores of 65.8 and 61.6 respectively. This would position both Weighted RETROFIT and RETROFIT with compound words and gloss words as second only to the top system.

V. DISCUSSION

Results on similarity judgment are mixed, although it should be noted that despite the fact that in principle average similarity appears to be a good measure of word relatedness, in our trials the maximum similarity between two words is a better predictor of human judgments on RG-65 with all algorithms. It's possible that in the absence of disambiguating context human judges are not actually good at combining the relatedness of different senses of words and instead specifically search for related meanings when evaluating similarity. It's worth noting that the metric by which our modifications provide the largest improvements is the metric which RETROFIT itself also performs best by. But, as discussed above and in [4], even

human judges often do not score particularly well similarity tasks, and in fact there may be no real "gold standard" on such a task.

The results of the synonym selection task are also mixed. On the ESL-50 dataset our modifications slightly underperform, while on the TOEFL dataset they provide an enormous improvement. We have not investigated the particulars of the datasets enough to see if there are anomalous features (over or under-representation of certain parts of speech, rare word senses, etc), or if these performance gaps are due more to the small sample size of the test data. Testing on a wider array of larger synonym selection datasets could yield insight into the models' shortcomings.

Our models are a noticeable improvement on WSD. Interestingly, the Weighted RETROFIT algorithm achieves the best scores on verbs across all metrics. Again, whether this is a quirk of the specific corpus is unclear. If not, it may indicate that homophonous verbs in English tend to be more distinct from each other than other parts of speech, perhaps because of more common metaphorical language use. We at least can say confidently that utilizing more features from WN is an across the board improvement.

FUTURE WORK

As mentioned above, the limited size and scope of the test sets leaves room for doubt about the models' performance on new datasets, especially when two datasets for the same task yield strikingly different results, like synonym selection. A useful exploration would be looking at domain-specific datasets and significantly larger datasets to identify which features of the models are most driving the performance.

We also use only a crude model of compound word vectors. An investigation of better compositional semantic models could greatly benefit the algorithm, as a large percentage of WN synsets contain compound words.

Our models are all trained on the relatively low dimensional global feature vectors produced by Huang et al [5], but significantly richer embeddings exist, such as the GoogleNews vectors, which are 300 dimensional and were trained on a 100 billion word corpus using CBOW [10]. We expect that the quality of the embeddings produced by the RETROFIT algorithms will scale with the quality of the underlying embeddings, and can hope for continual improvement as larger and better datasets become available.

REFERENCES

- [1] Agirre, E., & Soroa, A. (2009, March). Personalizing pagerank for word sense disambiguation. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 33-41). Association for Computational Linguistics.
- [2] Agirre, E., de Lacalle, O. L., & Soroa, A. (2014). Random walks for knowledge-based word sense disambiguation. *Computational Linguistics*, 40(1), 57-84.
- [3] Chen, X., Liu, Z., & Sun, M. (2014, October). A Unified Model for Word Sense Representation and Disambiguation. In *EMNLP* (pp. 1025-1035).
- [4] Faruqi, M., Tsvetkov, Y., Rastogi, P., & Dyer, C. (2016). Problems With Evaluation of Word Embeddings Using Word Similarity Tasks. arXiv preprint arXiv:1605.02276.
- [5] Huang, E. H., Socher, R., Manning, C. D., & Ng, A. Y. (2012, July). Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1* (pp. 873-882). Association for Computational Linguistics.
- [6] Jauhar, S. K., Dyer, C., & Hovy, E. (2015). Ontologically grounded multi-sense representation learning for semantic vector space models. In *Proc. NAACL* (Vol. 1).
- [7] Jufasky, D. & Martin, J. "Semantics with Dense Vectors". In *Speech and Language Processing, 3rd Ed.* (Draft of April 11, 2016)
- [8] Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2), 211.
- [9] Moro, A., & Navigli, R. (2015). SemEval-2015 task 13: multilingual all-words sense disambiguation and entity linking. *Proc. of SemEval-2015*.
- [10] Mihalcea, R. (1998). Sencor semantically tagged corpus. Unpublished manuscript.
- [11] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119)
- [12] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781..
- [13] Neelakantan, A., Shankar, J., Passos, A., & McCallum, A. (2015). Efficient non-parametric estimation of multiple embeddings per word in vector space. *arXiv preprint arXiv:1504.06654*.
- [14] Patwardhan, S., & Pedersen, T. (2006, April). Using WordNet-based context vectors to estimate the semantic relatedness of concepts. In *Proceedings of the EACL 2006 Workshop Making Sense of Sense-Bringing Computational Linguistics and Psycholinguistics Together* (Vol. 1501, pp. 1-8).
- [15] Princeton University (2010) "About WordNet." WordNet. Princeton University. <http://wordnet.princeton.edu>
- [16] Reisinger, J., & Mooney, R. J. (2010, June). Multi-prototype vector-space models of word meaning. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 109-117). Association for Computational Linguistics.
- [17] Rothe, S., & Schutze, H. (2015). Autoextend: Extending word embeddings to embeddings for synsets and lexemes. arXiv preprint arXiv:1507.01127.
- [18] Rubenstein, H., & Goodenough, J. B. (1965). Contextual correlates of synonymy. *Communications of the ACM*, 8(10), 627-633.
- [19] Turney, P. (2001). Mining the web for synonyms: PMI-IR versus LSA on TOEFL.
- [20] Weissenborn, D., Hennig, L., Xu, F., & Uszkoreit, H. (2015, July). Multi-objective optimization for the joint disambiguation of nouns and named entities. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing* (pp. 596-605).

Composition of Compound Nouns Using Distributional Semantics

Kyra Yee¹ and Dr. Jugal Kalita²

¹Pomona College

²University of Colorado, Colorado Springs

Abstract—The use of distributional semantics to represent the meaning of a single word has proven to be very effective, but there still is difficulty representing the meaning of larger constituents, such as a noun phrase. In general, it is unclear how to find a representation of phrases that preserves syntactic distinctions and the relationship between a compound’s constituents. This paper is an attempt to find the best representation of nominal compounds in Spanish and English, and evaluates the performance of different compositional models by using correlations with human similarity judgments and by using compositional representations as input into an SVM classifying the semantic relation between nouns within a compound. This paper also evaluates the utility of different function’s compositional representations, which give our model a slight advantage in accuracy over other state-of-the-art semantic relation classifiers.

Index Terms—compositional distributional semantics, nominal compounds, compound noun relation classification, Spanish word embeddings, nominal compounds in Spanish, SVM

I. INTRODUCTION

THE use of distributional semantics has become increasingly popular due to its effectiveness in a range of NLP tasks. The vector-based representation is computed by looking at the context of every instance of a specific word within a large corpus, which is based on the idea that the meaning of a word is determined by its associations with other words [17]. This idea has a theoretical backing in linguistics and psychology, since humans can often guess what a word means solely by looking at its context in a specific sentence [17]. Despite the success of vector-based representation in a wide variety on contexts, this method still has difficulty handling larger phrase structures and function words, as opposed to just isolated content words [13]. Vectors for larger phrases cannot be reliably used due to the sparseness of data [17]. Distributional semantics also has problems distinguishing parts of speech [12], [13]. In contrast, formal semantics is able to capture the syntactic structure of a phrase but lacks the ability to identify related words and ideas.

Recently there have been efforts to build compositional models that utilize distributional semantics. They have relied primarily on either adding or multiplying the vectors, which does a poor job of capturing the hierarchical, ordered structure of natural language [18], [13]. Ways of representing constituents larger than a single word that preserve the lexical and

syntactic function of a word in a phrase and best represent the relation between the constituents of a phrase is the next desired step in creating a more general and powerful framework for natural language semantics. One of the first steps towards that goal would be creating a more effective framework for representing and analyzing nominal compounds. Mitchell and Lapata [13], Mitchell and Lapata [12], and Guevara [18] have compared and empirically tested the effectiveness of different mathematical compositions in representing adjective-noun, verb-object, and noun-noun compounds, but there has been little research into representing nominal compounds that are longer than two words, and the vast majority of research has been in English, without cross-linguistic inquiries[12].

II. COMPOUNDING IN ENGLISH AND SPANISH

What constitutes a nominal compound is contested between linguists [2], [8]. For our purposes, we will use the definition given by Finin [8]:

A nominal compound is the concatenation of two or more nominal concepts which functions as a third nominal concept.

We will not consider the more theoretical qualifications for compounds nouns, such as structural fixity, and the exclusion of phrases with functional words, which are qualifications proposed by Moyna [2] for a more linguistically rigorous definition for Spanish compound nouns. The structure N N in English is productive, recursive, and compositional [3]. In Spanish, N N compounds are rarely productive, rarely contain more than two elements and are highly stylistic [3], [2].The process is that of lexical word-formation, as opposed to English,which has syntactic word-formation for N N compounds [3]. In Spanish, the creation of N N compounds more closely resembles the invention of a new morpheme,[3] which is reflected by the fact that only 2% of N N constructions are written as two words or a hyphenated word without one-word alternates [2]. Because of the limitations of N N constructions in Spanish, many consider the Spanish equivalent to the English N N structure to be the N P N structure, with a semantically empty preposition. This structure, similar to the English N N structure, is productive, recursive, and compositional [3].

Restricting our attention to compounds only consisting of two nouns in English, analyzing the meaning of nominal compounds computationally has proven to be a difficult task

K. Yee is participating in a Research Experience for Undergraduates (REU) with the Department of Computer Science, CO 80918 USA

because the listener must discern the relationship between the two words, which must be inferred contextually without any syntactic clues[8]. Consider the cases of “meeting room”, “salt water” and “aircraft engine”. “Room” defines the location for “meeting”, “engine” is a part of the “aircraft”, and “salt” is dissolved in “water”[8]. This problem of determining relations between the constituent nouns becomes even more difficult for longer phrases, because we now must determine the parse of the compound using contextual clues. In the phrase “computer science department”, “computer science” modifies “department”, instead of having “computer” modify “science department”. These factors pose challenges to vector-based representations of longer compound noun phrases.

In Spanish N P N constructions, despite the presence of a preposition or potentially determiners, is it still difficult to discern the relation between the constituent nouns. Spanish definite determiners are used in a much wider context than their English counterparts, so they do not provide much useful insight into the relation between the two nouns. In the majority of cases, the preposition is “de”, which is semantically empty in this construction[3], and is used to represent a multitude of relations, as seen from Table I (taken from Valle [1]).

English	Spanish	Meaning Implied
leather shoes	zapatos de piel	shoes made of leather
sports shoes	zapatos de deporte	shoes used to play sports with
winter shoes	zapatos de invierno	shoes to be worn in winter time
high-heel shoes	zapatos de tacón	shoes with high heels
display shoes	zapatos de muestra	shoes on display
Gucci shoes	zapatos de Gucci	shoes designed by Gucci

Table I
SPANISH SEMANTIC RELATIONS.

Thus the Spanish N P N construction poses similar challenges to the English N N construction. Our goal is to analyze compound nouns in English (which take on the form of N N) and semantically equivalent structures in Spanish, which take on the form N P N [4].

III. PREVIOUS WORK

A. Word Embeddings

A variety of methods for generating word embeddings have been proposed, most famously the GloVe and word2vec embeddings. The word2vec model, proposed by Mikolov and Dean [25], is a continuous skip-gram model that utilizes deep learning, which is able to capture precise syntactic and semantic word relationships to generate a vector representation of a word. The GloVe model [26] is a global bilinear regression model which combines the advantages of global matrix factorization and local context window methods. It utilizes statistical information by training on “non-zero elements in a word-word cooccurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus” [26]. The CW model, proposed by [23], implements a multilayer

neural network, where the first layer extracts features for each word and the second layer extracts features from a window of words. The model is refined using a supervised training step utilizing data from part-of-speech tagging, chunking, named entity recognition and semantic role labeling [23], [20]. The HPCA model [24] is generated by applying Hellinger PCA to a word co-occurrence matrix, which has the advantage of being much faster than training a neural net, and yields comparable results on a number of natural language processing tasks.

B. Compositional Models

Very little work has been done in distributional semantics for Spanish. Some studies have been done on the effectiveness of vector-based representations on Spanish [6], [7], but none have considered compositional models. Many studies have been done in English studying compositional models, [12], [13], [11], [14], [16], [18], [19], [15] but none have considered three or four word compound nouns.

There have been many functions suggested for how to compose two vectors. The general class of models representing the vector composition is defined by:

$$\mathbf{p} = f(\mathbf{u}, \mathbf{v}, R, K) \quad (1)$$

Where \mathbf{u} and \mathbf{v} are the constituent vectors, R represents their syntactic relation, and K represents any additional information required to interpret the semantics of \mathbf{p} . [12] Since we are only considering the composition of compound nominals, we can hold R fixed. We can also ignore K for simplicity, attempting to glean as accurate of a meaning as possible without further pragmatic context [12]. From these assumptions, we arrive at several more common potential functions: additive, multiplicative, and tensor product, respectively [12].

$$p_i = u_i + v_i \quad (2)$$

$$p_i = u_i \cdot v_i \quad (3)$$

$$p_{ij} = u_i \otimes v_j \quad (4)$$

The tensor product has interested some researchers since it does a better job of encoding syntactic information (the tensor product is not commutative, so it is seen as a representation that can distinguish “blood donor” from “donor blood”). However, the tensor product becomes very computationally expensive, as the number of dimensions grows exponentially as more constituents are composed [12], [15]. To account for this, lower dimensional approximations of tensors have been proposed [15]. The effectiveness of each of these equations, especially between the additive and multiplicative model, is still contested. There is evidence pointing to advantages of each model, so there is still much work to be done to determine the utility of each function for different applications in different contexts[16]. Another well-known function is the weighted additive function, which is regarded as being better at representing the syntactic relation between its constituents:

$$p_i = \alpha u_i + \beta v_i \quad (5)$$

With regard to nominal compounds, one study showed that the influence of the modifier noun has a much greater influence on the overall meaning of the compound than the head noun in German, with respect to both human ratings and vector-space models[14]. In contrast, another study determined that the semantic contribution of the modifier and head to a compound noun are approximately equal in English[10]. That being said, it could be the case that the average contribution of the modifier and head varies between languages, so determining a weighting for Spanish and English could yield different results to obtain the optimal weighted additive model. An extreme form of this formula would be to only use the vector from either the head or modifying noun:

$$p_i = u_i \quad (6)$$

$$p_i = v_i \quad (7)$$

It is also possible to combine the weighted additive and multiplicative model:

$$p_i = \alpha u_i + \beta v_i + \gamma u_i v_i \quad (8)$$

One major disadvantage to the multiplicative model is that the presence of a zero in either two component vectors will lead to a zero in the resulting vector, essentially meaning information from the noun-zero entries multiplied by zero was thrown away [12] Combining these two models could help alleviate that effect [12].

Other models for composition include utilizing a partial least squares regression [18] or using a recursive neural tensor network [19]. This paper will compare different models for the composition of two, three, and four word compounds in Spanish and English.

C. Automatic Compound Noun Interpretation

A variety of taxonomies have been proposed for the classification of compound noun relations, some of which consist of a relatively small number of semantic relations, while others propose an unbounded number [22]. The taxonomy created by Tratz and Hovy [22] has been widely used because of its comparatively high level of inter-annotator agreement for its relations and the large size of the data set. Kim and Baldwin[10] use wordnet similarity to classify a set of 2169 compounds into 20 semantic categories, achieving 53% accuracy. Girju [21] uses cross-linguistic data and an SVM model to achieve an accuracy of 77.9% on an unseen test set. Tratz and Hovy [22] use a dataset of 17509 compounds and a maximum entropy classifier to achieve 79.3% for cross-validation and 51% accuracy on an unseen test set using a set of 43 semantic relations, using wordnet, surface level, thesaurus based, and N-gram features. Girju et al.[27] uses an SVM to classify nominalized noun phrases. Verhoeven et al.[28] uses word embeddings to classify Dutch and Afrikaans compound nouns, achieving 47.8 % and 51.1%, respectively. Dima and Hinrichs [20] use a neural net on the concatenation of CW-50, FloVe-300, HPCA-200, and word2vec embeddings on the tratz dataset to achieve 77.7% accuracy on a ten-fold

cross-validation and 77.12% accuracy on an unseen test set. Although Dima and Hinrichs [20] and Verhoeven et al.[28] use word embeddings, none of the previously proposed models used the composition of word embeddings as input for their model.

IV. EXPERIMENTAL SETUP

A. Materials and Tools

We will evaluate the performance of each composition function in two ways: by analyzing its correlation with human similarity judgments, and also by seeing which composition function yields the best result for classifying compound noun relations using an SVM for English two-word compounds. For evaluating human judgments, we used the British National Corpus (<http://www.natcorp.ox.ac.uk/>) for English and the most recent wikidump from July 3, 2016 (<https://dumps.wikimedia.org/eswiki/latest/>) for Spanish to train word2vec embeddings. The WikiExtractor was used to extract and clean the wikidump (<https://github.com/attardi/wikiextractor>). The gensim package was used to extract 500 dimensional word2vec vectors, using the CBOW algorithm. For both corpora, stop-words were removed, and words that occurred less than 100 times for English and 50 times for Spanish were excluded from the model’s vocabulary. The Stanford POS tagger version 3.5.2 was used to extract Spanish nominal compounds,[5] and the BNC’s tagset was used to extract English compounds. Spanish and English compounds for the test set were randomly chosen from looking at the list of compounds that included one of the top 400 words that occurred in the most compounds. This was to ensure that for each compound in the test set, there would be a sufficient number of compounds that share one constituent word for comparison. There were six test sets: two, three and four word compounds for Spanish and English. For Spanish, this is with respect to nouns only, not counting the preposition or determiners when determining the length of the compound. 25 compounds were chosen for each test set, totaling up to 150 test compounds. For each word in the test compound, another two-word compound sharing that word was chosen for comparison. So for the four word-test sets, there were 100 pairs for comparison, for the three-word compound sets, there were 75 pairs for comparison, and for the two-word compound sets, there were 50 pairs for comparison.

For example, “periodo de expansión del imperio” was paired with “expansión del universo”, “embajador del imperio”, and “periodo de ausencia”. “Bomb squad chief” was paired with “bomb damage”, “drug squad”, and “police chief”.

The goal of analyzing compound noun relation classification is twofold; it will serve as another metric for comparing composition functions, and we will be able to determine if using the composition vectors as input to a classifier can improve overall performance of the classifier. We only perform this experiment in English for two-word compounds due to the availability of large preexisting annotated data sets. Verhoeven et al.[28] and Dima and Hinrichs [20] use word embeddings for semantic classification; however, they simply concatenate the embeddings for the constituent vectors as input. For

classifying semantic relations in English, we experimented with our BNC model, Google News Vectors (available at <https://code.google.com/archive/p/word2vec/>), GloVe vectors [26], CW vectors [23] and HPCA vectors [24]. For the utilization of word2vec vectors, we found better results with the Google News Vectors, probably due to the amount of data used to train them, so we report only classification results utilizing those here. For each embedding type, we chose the largest possible dimensions, since that has yielded the best results in [20]. Table II gives an overview of the information used to train each set. We used the dataset described in

Method	Embedding Size	Dictionary Size	Training Data Size	Support Corpora
word2vec	300	3,000,000	100.00 bn	Google News
GloVe	300	400,000	42.00 bn	Common Crawl
HPCA	200	178,080	1.65 bn	enWiki+Reuters+WSJ
CW	50	130,000	0.85 bn	enWiki+Reuters RCV1
word2vec	500	30,025	100 mn	BNC
word2vec	500	19,679	120 mn	esWiki

Table II
OVERVIEW OF DIFFERENT EMBEDDINGS

[29] (available at <http://www.isi.edu/publications/licensed-sw/fanparser/index.html>), which consists of 37 relations and 19,158 annotated compound nouns. Compounds with words that were not included in all of the different model embeddings' vocabularies were not included in the analysis, leaving a total of 18669 compounds. The set was partitioned into a training module that was 80% of the original set and a test set that was 20%. After experimenting with a variety of different classifiers and architectures, we used the Weka machine learning software (<https://weka.wikispaces.com/>) to implement an SVM with a polykernel, with feature selection using a gain ratio attribute evaluator and a ranker search. To create the input features, we concatenated the vectors for the constituent nouns and the composition function vector. We experimented with using the different word embeddings individually and in conjunction, and found the best results be concatenating the constituent and composition embeddings from the Google News word2vec, GloVe, HPCA, and CW sets, similar to the work done by Dima and Hinrichs[20].

B. Collecting Similarity Judgments

Responses were collected using Survey Gizmo (<https://www.surveygizmo.com/>), using unpaid volunteers. Subjects were asked to rate how similar or dissimilar compound noun pairs were on a Likert scale. Each pair was presented twice, once as "compound 1, compound 2" and again as "compound 2, compound 1" to account for an asymmetry of human judgments. Pairs were presented in random order. Surveys were self-paced and took approximately fifteen minutes. For the English survey, there were 7 participants. For the Spanish survey, there were 4 participants. Participants ranged in age from 15-55, and were self-reportedly fluent in the language of the survey. One future direction of this study will be to gather more data from more people, especially for Spanish. For each pair, the average

similarity was calculated on a scale of 1 to 5, 5 being most similar and 1 being the most dissimilar. Ratings from each participant were averaged to use to correlate with the model's cosine similarity predictions.

C. Composition Methods

For the human judgment correlation task, for each compound in the test and comparison set, representations were generated by taking the vector representations from the word2vec model using the CBOV algorithm trained from the BNC and esWiki corpora. For entries in the Spanish test set, only the nouns were considered for composing the phrase. Since the preposition is largely semantically empty and only serves to illustrate the syntactic connection between the nouns, it is ignored. As we have previously seen, the preposition "de" encodes a wide variety of semantic relations; however, there is a minority of nominal compounds that use different prepositions like "por", "para", "entre", etc. We will naively assume here that the preposition does not encode semantic information and focus only on compounds using the most common preposition "de", which is a bit of a generalization. A future direction of study would be to incorporate information from the preposition into the composition vector. Articles were also ignored, since they also do not provide much semantic meaning, especially considering their more generalized usage in Spanish compared to English. The composition of the constituent words for each compound was then calculated using the following functions: simple additive (equation (2)), multiplicative (equation (3)), tensor product (equation (4)), head only (equation (7) for English, (6) for Spanish), modifier only (equation (6) for English, (7) for Spanish), weighted additive (equation (5)), and combined weighted additive and multiplicative (equation (8)). For three word compounds, data was parsed by hand into (n1 n2) n3 or n1 (n2 n3) so that syntactically sensitive functions could be properly applied recursively. The same method was applied to four-word compounds. For compounds longer than two words, the head only and modifier only models were not calculated, since there are multiple modifiers and heads.

D. Determining the Parameters of the Weighted Additive and Combined Models

The parameters of the weighted additive model were determined in two different ways. First, we considered nine models, with weights varying from 0.1 to 0.9 in a step size of 0.1, where the sum of α and β adds to one, where the model with the highest correlation to the human judgments was taken as optimal. For the purposes of this experiment, the magnitude of the vector does not matter, because the cosine similarity is taken for the final metric, which does not take magnitude into account. We used a grid search to find the optimal values for α and β , but without the constraint that they had to add to one, again maximizing the correlation to human judgments. Likewise, for the combined model, we used a similar grid search, without the traditional constraint. The model parameters are described in Table III, where NWA stands for normalized weighted additive and OWA stands for optimized weighted additive.

	Combined Model			OWA		NWA	
	α	β	γ	α	β	α	β
two-word Spanish	0.099	0.101	0.000	0.098	0.098	0.5	0.5
two-word English	0.267	0.264	9.697	0.874	0.898	0.5	0.5
three-word Spanish	1.452	1.943	-0.006	1.333	1.749	0.2	0.8
three-word English	0.842	0.724	0.000	0.821	0.719	0.9	0.1
four-word Spanish	0.949	1.387	4.422	1.065	1.639	0.1	0.9
four-word English	0.939	0.869	2.580	0.927	0.869	0.1	0.9

Table III
PARAMETERS FOR THE COMBINED, OPTIMIZED WEIGHTED ADDITIVE,
AND NORMALIZED WEIGHTED ADDITIVE MODELS

In Spanish, the head is the first noun, and would be weighted with α , whereas the head is the second noun in English, and would be weighted with β . So we see that heavily weighting the modifier is a consistent trend across the combined, normalized additive, and optimized additive models in English and Spanish for compounds longer than two words, with the exception of the four-word normalized additive English set. This inconsistency could be due to idiosyncrasies in the relatively small data set. For two-word compounds in English and Spanish, an even weight distribution yielded the best results. This could imply that as the length of the compound noun grows, the semantic importance of the modifier increases.

V. EVALUATION

For the human similarity judgments, we calculated inter-subject agreement using Spearman’s ρ , using leave-out one resampling as employed by [13], with the results given in Table IV.

2W Spanish	2W English	3W Spanish	3W English	4W Spanish	4W English
0.341	0.441	0.357	0.347	0.170	0.321

Table IV
INTERSUBJECT AGREEMENT FOR HUMAN SIMILARITY JUDGMENTS

For future research, we hope to gather more data from Spanish-speakers to help improve the accuracy of the data. For the two-word English set, we see that the similarity judgment is consistent with previous work [13]. As a general trend, inter-subject agreement declines as the compounds get longer.

We evaluated the similarity of two compounds by taking the cosine of their vectors, a commonly used metric [12]. To test if a composition model’s results were consistent with human judgments, we used Spearman’s correlation, where we compared the cosine with the average human similarity judgment. Similar to [12], the results indicate that the similarity judgment task was relatively difficult, but there still was a descent amount of consistency between participants.

For noun relation classification, we used two metrics. We performed a ten-fold cross-validation on the training set, and also tested each model on the unseen test set. For the parameterized functions, we used the optimized parameter

values from the corresponding human judgment correlation test. Since the optimal normalized parameters from the 2-word English set was 0.5 and 0.5, we did not perform a test for the normalized weighted additive set, since the proportions are the same as the simplified additive model. We also did not test the tensor product model, due to constraints in dimensionality.

VI. RESULTS

A. Correlation with Human Similarity Judgments

Table V shows the model’s predictions correlated with the human judgment using Spearman’s ρ .

	2W Span	2W Eng	3W Span	3W Eng	4W Span	4W Eng
simple additive	0.365	0.617	0.585	0.331	0.230	0.650
multiplicative	0.258	0.624	0.227	-0.057	0.105	0.372
tensor	0.357	0.621	0.040	-0.041	0.266	0.321
head	0.280	0.443				
modifier	0.191	0.060				
normalized weighted additive	0.365	0.617	0.521	0.312	0.289	0.336
optimized weighted additive	0.371	0.633	0.690	0.330	0.435	0.654
optimized combined	0.342	0.670	0.652	0.338	0.434	0.658

Table V
SPEARMAN’S CORRELATION BETWEEN HUMAN SIMILARITY JUDGMENTS
AND COSINE SIMILARITY PREDICTIONS

The simple additive model and the multiplicative model yield comparable results for two-word compounds, but the effectiveness of the multiplicative model declines for longer compounds. This could be due to the previously discussed fact that zero or low-valued entries in the vector can essentially “throw away” data in the component vector, leading to poor results as more vectors are composed. As more and more vectors are composed, this problem is exacerbated and begins to affect performance. Likewise, the tensor product performs well on two-word compounds in comparison with the additive model, but less so on longer compounds, especially three-word compounds. This may imply that in addition to dimensionality challenges, the tensor product may face similar limitations to the multiplicative model for composing larger phrases. For both English and Spanish, for two-word compounds, the head-only model outperforms the modifier-only model. All other models outperform the head-only and modifier-only models, indicating the utility of the composition functions. For the optimized weighted additive and combined models, the results are very comparable, with the optimized additive model slightly outperforming the normalized additive model. The combined and weighted additive models yield the most promising results, especially since their accuracy is relatively consistent for handling longer phrases.

B. Compound Noun Relation Classification

	word2vec+HPCA+CW+GloVe		word2vec		GloVe		HPCA		CW	
	CV	test set	CV	test set	CV	test set	CV	test set	CV	test set
no composition function	76.76	77.3	75.41	76.67	73.35	73.21	71.60	72.39	61.96	62.26
simple additive	76.52	76.95	75.85	76.01	72.63	73.59	71.36	71.59	61.98	62.23
weighted additive	76.47	77.70	74.80	76.76	72.70	73.62	71.37	71.48	62.02	62.31
multiplicative	78.78	78.23	75.82	76.04	73.42	73.30	71.95	72.50	62.52	62.58
combined	78.69	78.09	75.99	76.09	73.38	73.30	71.36	71.59	62.27	62.18

Table VI

CROSS-VALIDATION ACCURACY AND ACCURACY ON AN UNSEEN TEST SET FOR SEMANTIC RELATION CLASSIFICATION

Table VI gives the results for each tested function on the different word embeddings, including the concatenation of all the different embeddings. The CV column represents the 10-fold cross-validation accuracy, and the test set is comprised of unseen noun compounds. Input with only the constituent vector embeddings without the composition function was also tested to give a baseline. Adding the composition function improves the performances for every type of embedding, with the most dramatic improvement in the concatenated word2vec+HPCA+CW+GloVe model.

We achieved the best results using the concatenation of the word2vec, HPCA, CW, and GloVe embeddings. Adding the composition function improves this models performance by as much as 2.02% using the multiplicative function, demonstrating the utility of using a compositional function during classification. The simple additive and weighted additive models actually perform worse in cross-validation than using no composition function at all. The combined models γ parameter was 9.697, so the multiplicative component of the combined model mostly overpowers the additive components, which explains why its performance is similar to that of the multiplicative model. Our model slightly outperforms Dima and Hinrich [20], with its high cross-validation score being 77.7%, and is comparable to the state of the art model of Tratz and Hovy[22], achieving 79.3%. However, the model of Tratz and Hovy[22] only achieves 51% accuracy on an unseen test set, whereas our model is much more consistent, with 78.23% accuracy. Again, we narrowly outperform Dima and Hinrich [20], with its accuracy on an unseen test set, which was 77.12% [20]. Tratz and Hovy [22] use a slightly different set of relations and data set, but similar to the work of Dima and Hinrichs[20], the consistency when testing unseen compounds points to the robustness of our model. It is also clear that the small performance increase spurred by the addition of the composition function gives our model its slight increase in accuracy over the model of Dima and Hinrichs[20], with a 4.84% decrease in relative error for cross-validation and 4.85% decrease in relative error for an unseen test set.

VII. DISCUSSION

With regards to the effectiveness of the additive and multiplicative classes of models, which has been contested by different sources, this paper presents strong evidence that multiplicative class models do not perform well for longer phrases. This idea is further supported by the low γ parameters in the optimized combined model for three and four word

compounds. However, within the context of semantic relation classification, the multiplicative model is the strongest, whereas the additive model does not improve performance significantly, and sometimes even worsens performance. One interesting direction of future study would be to see which function performs best for classifying longer compounds, since the multiplicative model did not perform well for the human similarity correlation task for longer compounds. This paper also suggests that the semantic importance of the head noun diminishes as the compound gets longer, and that the semantic importance of the modifier becomes greater, as illustrated by the optimized parameters of the weighted additive models. One future direction of study would be to implement more complex composition functions, or to incorporate information from the prepositions in Spanish compound nouns into the composition vector. Another direction of study would be to expand the noun relation classification task to a Spanish data set, and compare results, or to expand the classification task to three or four word compounds in English. This study points to the robustness of the combined model, since it is able to capture information from both the additive and multiplicative models. It performs well for three and four word compound human judgment similarity correlation, and it performs well in the relation classification task. The flexibility of its parameters, which can vary between languages and for compound nouns differing in length, makes it very promising.

VIII. CONCLUSION

The goal of this research is to find the optimal way to represent compound nouns of length two or greater using a vector-based representation. We have illustrated the utility of the multiplicative model in relation classification, but it has shortcomings in representing larger phrases in comparison to the additive class of models. Our new classification system, which incorporates composition vectors into SVMs, is comparable to other state-of-the-art models using cross-validation, or slightly outperforms them using an unseen test set.

REFERENCES

- [1] Valle, Ana. "On the teachability of nominal compounds to Spanish learners of English for specific purposes." *English for Specific Purposes: Studies for Classroom Development and Implementation*. UCA Servicio de publicaciones (2007): 73-97.
- [2] Moyna, María Irene. *Compound words in Spanish: theory and history*. Vol. 316. John Benjamins Publishing, 2011.24-61.
- [3] Bauke, Leah S. *Symmetry breaking in syntax and the lexicon*. Vol. 216. John Benjamins Publishing Company, 2014.21-26.
- [4] Girju, Roxana. "The syntax and semantics of prepositions in the task of automatic interpretation of nominal phrases and compounds: A cross-linguistic study." *Computational Linguistics* 35.2 (2009): 185-228.

- [5] Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003, pp. 252-259.
- [6] Etcheverry, Mathias and Wonsaver, Dina "Spanish Word Vectors From Wikipedia." Proceedings of Language Resources and Evaluation. (2016): 3681-3685.
- [7] Al-Rfou, Rami, Bryan Perozzi, and Steven Skiena. "Polyglot: Distributed word representations for multilingual nlp." (2013).
- [8] Finin, Timothy Wilking. "The semantic interpretation of compound nominals." (1980). PhD Thesis, University of Illinois, Urbana-Champaign. 2-5, 89-120
- [9] Kim, Su Nam, and Timothy Baldwin. "Interpreting semantic relations in noun compounds via verb semantics." Proceedings of the COLING/ACL on Main conference poster sessions. Association for Computational Linguistics, 2006.
- [10] Kim, Su Nam, and Timothy Baldwin. "Automatic interpretation of noun compounds using WordNet similarity." Natural Language Processing-IJCNLP 2005. Springer Berlin Heidelberg, 2005. 945-956.
- [11] Reddy, Siva, et al. "Dynamic and Static Prototype Vectors for Semantic Composition." IJCNLP. 2011. 705-713
- [12] Mitchell, Jeff, and Mirella Lapata. "Composition in distributional models of semantics." Cognitive science 34.8 (2010): 1388-1429.
- [13] Mitchell, Jeff, and Mirella Lapata. "Vector-based Models of Semantic Composition." ACL. 2008. 236-244
- [14] Im Walde, Sabine Schulte, Stefan Miller, and Stephen Roller. "Exploring vector space models to predict the compositionality of German noun-noun compounds." Proceedings of the 2nd Joint Conference on Lexical and Computational Semantics. 2013. 255-265.
- [15] Stephen Clark, Tamara Polajnar Luana Fagarasan. "Reducing Dimensions of Tensors in Type-Driven Distributional Semantics." 2014. 1036-1046
- [16] Baroni, Marco, and Roberto Zamparelli. "Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space." Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2010. 1183-1193.
- [17] Erk, Katrin. "Vector space models of word meaning and phrase meaning: A survey." Language and Linguistics Compass 6.10 (2012): 635-653.
- [18] Guevara, Emiliano. "A regression model of adjective-noun compositionality in distributional semantics." Proceedings of the 2010 Workshop on Geometrical Models of Natural Language Semantics. Association for Computational Linguistics, 2010. 33-37
- [19] Socher, Richard, et al. "Semantic compositionality through recursive matrix-vector spaces." Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics, 2012. 1201-1211.
- [20] Dima, Corina, and Erhard Hinrichs. "Automatic Noun Compound Interpretation using Deep Neural Networks and Word Embeddings." IWCS 2015 (2015): 173-183.
- [21] Girju, Roxana. "Improving the interpretation of noun phrases with cross-linguistic information." Annual Association for Computational Linguistics. Vol. 45. No. 1. 2007.568-575.
- [22] Tratz, Stephen, and Eduard Hovy. "A taxonomy, dataset, and classifier for automatic noun compound interpretation." Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2010. 678-687.
- [23] Collobert, Ronan, et al. "Natural language processing (almost) from scratch." Journal of Machine Learning Research 12.Aug (2011): 2493-2537.
- [24] Lebre, Rémi, and Ronan Collobert. "Word embeddings through hellinger PCA." (2013).
- [25] Mikolov, T., and J. Dean. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems (2013). 1-9.
- [26] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.
- [27] Girju, Roxana, et al. "Support vector machines applied to the classification of semantic relations in nominalized noun phrases." Proceedings of the HLT-NAACL Workshop on Computational Lexical Semantics. Association for Computational Linguistics, 2004. 73-78.
- [28] Verhoeven, Ben, Walter Daelemans, and Gerhard B. Van Huyssteen. "Classification of noun-noun compound semantics in Dutch and Afrikaans." Proceedings of the Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa (PRASA 2012). 2012.
- [29] Tratz, Stephen. Semantically-enriched parsing for natural language understanding. University of Southern California, 2011.

Computation Pattern Classification for Compiler Optimization

Nathan Harmon*, Qing Yi[†] and Jugal Kalita[‡]

University Of Colorado

at Colorado Springs

Colorado Springs, Colorado 80918

Email: *nharmon@uccs.edu, [†]qyi@uccs.edu, [‡]jkalita@uccs.edu

Abstract—The first step towards automated code optimization is program comprehension. Developers can manually identify computation patterns, but it consumes more than half of their time [2]. Once the compiler knows the computation model of a program, optimizations can be applied that are likely to perform well. We present an extensible machine learning approach using dynamic features to determine the computation patterns with 94% accuracy.

I. INTRODUCTION

Different computation models require different types of optimizations. Stencil code requires vastly different optimizations than Sparse Matrix code. In the pursuit of automated optimization, extracting computation patterns such as categories of code (Stencil, Sparse Matrix, etc.) is the first step. Comprehension of the code can be done by the developer, but consumes more than half of the developers time [2]. The introduction of the human element can also introduce additional errors from incorrectly tagged code. Once the type of computation is identified to a compiler, it can select optimizations that will likely be effective. Code comprehension is the first of many steps towards automated optimization of code.

Machine learning has been used to solve many difficult problems. Before machine learning can be used to address a problem, the problem must first be constructed in a way that machine learning is suited to solve, such as classification. To exploit the power of machine learning, we formulated the problem as a classification problem. We construct a finite number of classes representing different computation patterns and an Open Set for all other computation patterns. For the case study, we focus on separating various types of Stencil code. The features were extracted at runtime, as a time series. The uses of dynamic features has not been explored in previous work. We then use a Dynamic Recurrent Neural Network (LSTM) to classify different types of stencils. We must use dynamic networks since the length of the features extracted from the programs varies for each program. We will show that this approach has promise for future work in program compression.

II. RELATED WORK

Machine learning has been used to improve compiler optimization. Much of the past work is focused on heuristics to improve the selection of optimization. Heuristics created by

hand or typically designed to operate on a particular platform and do not generalize to other platforms. Machine learning allows the heuristics to be optimized for each platform, so it selecting better optimizations for that platform.

Monsifrot, et al. [5] used decision trees and boosting to create new compiler heuristics for loop unrolling. The features were the number of statements, number of arithmetic operations, minimum number of iterations, number of array accesses, the amount of array element reuses and the number of if statements. Using the features they constructed a tree with the probability of loop unrolling speeding up the computation. The decision tree was used to replace the existing heuristic.

Stephenson et al. [6] used genetic programming to learn priority functions – cost function describing the efficacy of a heuristic. The priority function must be learned for each application. The genetic algorithm creates a priority function that selects the heuristic that will choose the best optimization for the application. They show that new heuristic can be created from the result of the genetic algorithm.

Cavazos, et al. [1] also used machine learning (logistic regression) to select the best optimizations at compile time. Rather than using static features, as most others works did, they used performance counters as inputs to the model. They determined that the performance counters gave more information on the behavior of a program than static code features.

Fursin et al. [3] created a machine learning based compiler. Which reduced the number of iterative compilation, needed to select good optimizations for unknown platforms. They used static program features (variables, types, instructions, basic blocks, temporary variables, etc.) along with runtime behavior to train various machine learning models.

III. FEATURE SELECTION

Many past approaches used static features including variable names, comments, and documentation. Variable names and comments contain meaningful information, yet they depend on the programmer. They do not generalize well across programs, reducing their usefulness as features. In contrast, we collect runtime features and only characteristics that are present in all programs. The goal of the feature is to encapsulate the inherent pattern in the computation. We do this by focusing on the memory reference in the program. For each memory reference, we create a feature containing six elements. The

items are: read or write, data type, dependents, modification, scope, and relative distance from last access (arrays only).

A. Read or Write

Knowing whether the reference is being read or written to, is essential for understanding what role a reference plays in the computation. If a memory reference is being read, it indicates that the value will be used to calculate a value yet to be seen. Whereas a write is the result of computation.

B. Data type

Secondly, the data type. The data type limits how a memory reference is used, which in turn limits how the reference is used in the computation. Often structures such as loops can be identified by examining the alternation is the data types.

C. Dependents

Variable dependents lends information about the computation while remaining general across different types of computation.

D. Modification

We define modification as any arithmetic operation (+, -, /, *). The modification directly captures the computation that is occurring.

E. Scope

The scope is crucial since it affects what can be accessed. The representation of the scope is only the changes in the scope, not a particular block. Each time the scope changes a boolean is negated, creating an alteration of 0 and 1. The change in scope generalizes better across different programs, then indicating a particular block.

F. Relative Distance

If the reference was to an array, then the relative distance from the last access to that array is included in the feature. The relative offset captures the array access pattern.

IV. FEATURE CREATION

The programs are instrumented to output the above features at runtime. The test programs for this paper are input independent. If the program is input dependent, then it should be run multiple time with different inputs to collect a variety of possible outputs. The features that are output by the programs are time series data. Each feature by itself includes limited information about the computation, but the sequence of features encapsulates essential characteristics of the computation.

V. DATA SET

The data set consists of 9565 unique generated stencil codes. The stencil code contains 1-point through 11-point stencils. Each of the stencils was instrumented to output the features. Then run and the features were collected.

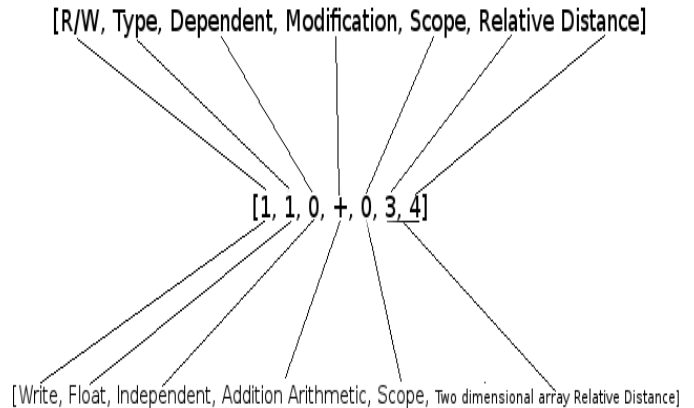


Fig. 1. Example Feature

VI. PROGRAM CLASSIFICATION

To take advantage of the power of machine learning, we constructed a classification problem. There are an infinite number of possible computation patterns, so the classes will never cover all possibilities. With that said it is feasible to cover all possibilities that have known optimizations. In this case study, we will separate stencil code into classes. The classes will represent the number of points in each stencil. There will also be an Open Set for stencil code that we do not have a class for. Since we have a good sampling of the types of stencil code that belong in the Open Set, we did not do any additional work in the Open Set filed.

VII. NEURAL NETWORK

Since the features make variable length time series, we use a standard implementation of a Dynamic Recurrent Neural Network (LSTM)[4]. We use a dynamic version of the networks since the number of features from each program varies. The numbers that represent each element in are features do not hold meaning, so we found that we obtained better results when we used learned embeddings of the features. The features are input to the network using one-hot encoding, and the learned embedding is looked up for the feature. The current implementation requires a minimum of 25 features in each series and tunicates the series if more than 100 features exist. The network has six outputs; the first five outputs are for a 1-point stencil, 8-point stencil, 9-point stencil, 10-point stencil and 11 point stencil. These are the most common types of stencil code in the test set. The network has also been evaluated with different configurations of output classes with

will be done with additional types of computation outside of stencil code.

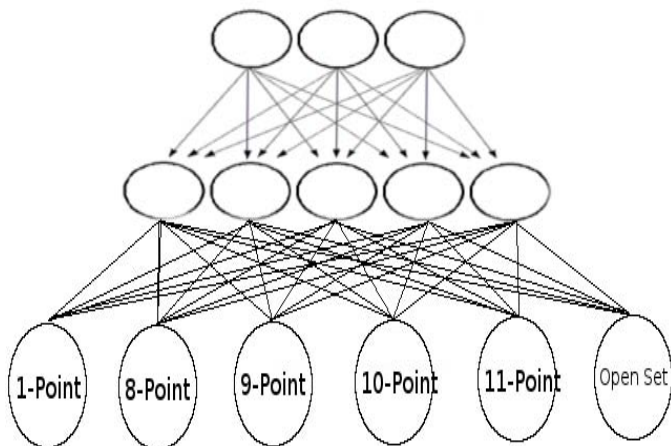


Fig. 2. Network Outputs

similar results. The last class is an Open Set for all other types of stencils.

TABLE I
NETWORK RESULTS

Learned Embeddings	Accuracy
No	74%
Yes	94%

VIII. EVALUATION

We evaluated the network using tenfold validation across the data set. Without the learned embeddings, the accuracy of the network was 74%. Next, we tested the network using learned embeddings which obtaining 94% accuracy. Changing which classes of stencil code we classified and which belonged to the Open Set did not change the accuracy. This result shows promise of this being a valid approach to identifying computation patterns in code. For this case study, we had good training data for the types of computation that belonged in the Open Set. In an actual application, we would not have good examples of what belongs in the Open Set. When randomly generated features were input into to the network to simulate unseen data, which should be classified into the Open Set, we saw poor accuracy. This is expected since there was no designed to ensure the unseen data would be classified as Open Set.

IX. FUTURE WORK

Additional effort will be put into classifying unseen computation patterns as Open Set. This will make this approach valid for use in the selection of optimization. More evaluation

REFERENCES

- [1] J. Cavazos, G. Fursin, F. Agakov, E. Bonilla, M. F. O’Boyle, and O. Temam. Rapidly selecting good compiler optimizations using performance counters. In *International Symposium on Code Generation and Optimization (CGO’07)*, pages 185–197. IEEE, 2007.
- [2] T. A. Corbi. Program understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.
- [3] G. Fursin, Y. Kashnikov, A. W. Memon, Z. Chamski, O. Temam, M. Namolaru, E. Yom-Tov, B. Mendelson, A. Zaks, E. Courtois, et al. Milepost gcc: Machine learning enabled self-tuning compiler. *International journal of parallel programming*, 39(3):296–327, 2011.
- [4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [5] A. Monsifrot, F. Bodin, and R. Quiniou. A machine learning approach to automatic production of compiler heuristics. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 41–50. Springer, 2002.
- [6] M. Stephenson, S. Amarasinghe, M. Martin, and U.-M. O’Reilly. Meta optimization: improving compiler heuristics with machine learning. In *ACM SIGPLAN Notices*, volume 38, pages 77–90. ACM, 2003.

Automatic Algorithm Selection in Computational Software Using Machine Learning

Matthew C. Simpson

NC State University

Raleigh, NC 27695

Email: mcsimps2@ncsu.edu

Qing Yi

University of Colorado at Colorado Springs

Colorado Springs, CO 80918

Email: qyi@uccs.edu

Jugal Kalita

University of Colorado at Colorado Springs

Colorado Springs, CO 80918

Email: jkalita@uccs.edu

Abstract—Computational software programs, such as Maple and Mathematica, heavily rely on superfunctions and meta-algorithms to select the optimal algorithm for a given task. These meta-algorithms may require intensive mathematical proof to formulate, incur large computational overhead, or fail to consistently select the best algorithm. Machine learning demonstrates a promising alternative for automatic algorithm selection by easing the design process and overhead while also attaining high accuracy in selection. Two case studies are selected to demonstrate this hypothesis, namely the resultant superfunction, which computes the pairwise difference between roots of any two polynomials, and the shortest tour superfunction, which finds the least weight Hamiltonian cycle in a graph. These functions have multiple algorithms available for their computation in many mathematical software programs. Neural networks, random forests, k-nearest neighbors, and linear and RBF kernel SVMs are each trained as automatic algorithm selection tools. For the resultant superfunction, the models are trained to select algorithms based on which will give the lowest runtime for a given input. In this case, neural networks perform the best, correctly selecting the optimal algorithm out of the four available 86% of the time in Maple and 78% of the time in Mathematica. When used as a replacement for pre-existing meta-algorithms, the neural network brings about a 68% runtime improvement in Maple and a 49% improvement in Mathematica. For the shortest tour superfunction, the machine learning models are trained to minimize both runtime and approximation error subject to user specified weights on how important each aspect of performance is. Random forests outperform other machine models, attaining a 99% accuracy in selecting the optimal algorithm. When Mathematica’s meta-algorithm is replaced with the random forest model, not only is the model able to select the algorithms that give the shortest tour (zero approximation error) in a given graph, but it does so while lowering the runtime by 75% compared to Mathematica’s meta-algorithm.

Index Terms—Resultant, Shortest Tour, Traveling Salesman Problem, Machine Learning, Meta-algorithms, Superfunctions, Algorithm Selection, AAS, Maple, Mathematica, Computational, Software

I. INTRODUCTION

The algorithm selection problem was first formalized by Rice [1] and is stated as follows:

Given the space of all problems \mathcal{P} , along with an algorithm space \mathcal{A} which contains all known algorithms to solve the problems in \mathcal{P} , determine a selection mapping $S : \mathcal{P} \rightarrow \mathcal{A}$ that maximizes performance for each problem $x \in \mathcal{P}$.

That is, if we measure performance in \mathbb{R}^n , where there are n dimensions of performance to take into consideration (e.g.

runtime, memory usage, error), and if we define a performance measure $p : \mathcal{A} \times \mathcal{P} \rightarrow \mathbb{R}^n$ that maps an algorithm applied to a problem instance to its performance in \mathbb{R}^n , we wish to find the aforementioned selection mapping S such that for any $x \in \mathcal{P}$, it holds that $\|p(S(x), x)\| \geq \|p(a, x)\|$ for each $a \in \mathcal{A}$. In essence, S finds the algorithm that offers the best performance for any problem instance.

A robust solution to the algorithm selection problem is especially important for NP-Hard problems, where the algorithm runtimes can be highly variable based on the inputs to the problem. Mathematical and scientific computational software, such as Matlab, Mathematica, Maple, Sage, and NumPy, have largely resorted to employing superfunctions and meta-algorithms as a solution to this problem. Superfunctions are methods that encapsulate function calls to more specific methods to compute what the user desires. For example, the superfunction *dsolve* can be called by a Maple user to solve a system of ordinary differential equations, but it does so by, in turn, calling more specific subroutines that are available to solve such systems, such as the Taylor series method or the Rosenbrock method. To determine which specific method to call, these superfunctions use a meta-algorithm, which is responsible for making an educated choice as to which algorithm will perform the best, usually with regards to runtime.

Algorithm selection ultimately relies on properties of the inputs. Some inputs simply just won’t work with certain algorithms, and other algorithms are able to provide performance enhancements and error reductions if the input is well conditioned. The latter opens up an opportunity for machine learning to be the ultimate selector of which algorithm to employ. Meta-algorithms rely on preconceived notions and rules of thumb about which algorithm should be the best in a given situation, rather than on statistical data about what has been the best approach. Designing meta-algorithms can be extremely complicated, especially to take in as many features of the inputs as possible. It is very likely some features are missed or will have to be ignored for the sake of computation. The difficulty of design is escalated when multiple aspects of the output are important, such as when both runtime and error reduction need to be taken into account instead of just one or the other. In addition, the complicated design of meta-algorithms can add large overhead to what may seem to be a

simple task. Sometimes, this overhead is more than that of the ultimate algorithm selected [2].

Machine learning can be used in practice to replace meta-algorithms by acting as a classifier to analyze important features of the input and then classify the input into which algorithm would be appropriate for it. This approach allows the program to make use of a wider feature set to make more precise decisions, compared to the smaller feature set the designers of meta-algorithms are usually forced to focus on. This added precision brings about a higher accuracy in the proportion of times the best algorithm is selected. Machine learning also avoids the need to create rules of thumb based on each feature and eases the design process needed to create an algorithm selection tool. Such an approach can be applied to any superfunction with performance dependence on inputs. For each superfunction, the important input features that affect performance need to be extracted to train the model, a general process which does not require significant tailoring to any specific problem as meta-algorithms generally do. Additionally, the problem of adding newly implemented algorithms to the pool of ones to select from becomes trivial; the machine learning model need only be retrained to account for these new algorithms, which takes a matter of seconds, compared to a total rewrite and overhauled design of meta-algorithm code. Lastly, machine learning can take into account multiple facets of runtime when choosing an algorithm, such as runtime, memory usage, and error, whereas meta-algorithms typically have to focus on only one of these aspects.

This paper aims to make headway by using machine learning as a tool for automatic algorithm selection (AAS) in computational software. Two case studies are used to provide evidence for this hypothesis: the resultant superfunction and the shortest tour superfunction. The resultant superfunction is available in most symbolic computation programs (*e.g.* Mathematica, Maple, Sage), and others with greater support for graph objects typically have a shortest tour function implemented (*e.g.* Mathematica, SAS). For these case studies, Maple and Mathematica are used to evaluate the results of machine learning and compare them to their corresponding meta-algorithm implementations. The main technical contributions of this paper are

- Developing a general approach and necessary formulations for using machine learning to automate the selection of algorithms in computational software
- Demonstrating, through the resultant and shortest tour superfunctions, the success of machine learning as a tool for AAS
- Empirically comparing the performance of machine learning models to that of Maple's and Mathematica's meta-algorithms

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the mathematical background of the resultant and the algorithms used for its computation. Maple's meta-algorithm for choosing amongst these algorithms is also discussed, as well as the features and

datasets used for training multiple machine learning models. Section 4 does the same for the shortest tour superfunction. Section 5 evaluates the accuracy of different machine learning approaches and compares these results to those of Maple and Mathematica, the main ideas of which are again summarized in Section 6.

II. RELATED WORK

Meta-algorithms are not the only approach to the algorithm selection problem. The algorithm portfolio paradigm [3] was an early attempt at a solution, which, in effect, selects a portion of all available algorithms and runs them in parallel until one of them finishes. The parallel computation causes a large resulting overhead, but nevertheless, this method shines for certain problem classes with heavy-tailed runtime distributions. In these cases, the algorithm portfolio paradigm is more advantageous than running a single algorithm [4]. Dynamic algorithm portfolios [5] provide a step forward from traditional algorithm portfolios by running a set of algorithm in parallel, but then iteratively updating the priority of each process by how well each is performing. The biggest problem here, however, is developing and implementing a measure in each algorithm that allows one to judge the current progress made. More importantly, this measure needs to be designed such that it allows for a fair comparison between different algorithms.

For many end users in scientific and mathematical software, these parallel computing approaches are not feasible given their large overhead. Although these approaches minimize runtime, no regard is given to the resource management aspect of performance. Because of this, meta-algorithms have been selected as the preferred tool in computational software.

The idea of using statistical and machine learning techniques for the selection of algorithms is not unheard of. Brewer [6][7] brought about the idea of using regression for performance predictions by using linear fitting to predict the runtime of different implementations of multiprocessor libraries on unseen architectures. In the field of meta-learning, Bradzil et al. [8] applied this technique using an Instance-Based Learning approach to select appropriate learning algorithms for different sets of problems. Similarly, the study in this paper continues to build upon this idea in the context of computational software.

III. THE RESULTANT CASE STUDY

To demonstrate the overarching hypothesis that machine learning can be a better alternative to meta-algorithms for use in computational software, a popular superfunction was chosen as the first case study, namely the resultant. The resultant is a fundamental tool used in computer algebra, algebraic geometry, algebraic cryptography, and elimination theory, and it finds higher level use in algorithms for integration, solving systems of nonlinear equations, computing the discriminant of two polynomials, analyzing greatest common divisors, and so forth.

A. The Resultant

Given two polynomials $a, b \in \mathbb{F}[x_1, x_2, \dots, x_k]$, where \mathbb{F} denotes an integral domain, of degree n and m respectively, they can be written with respect to the variable x_l , $1 \leq l \leq k$, as $a(x_l) = a_n x_l^n + \dots + a_1 x_l + a_0$ and $b(x_l) = b_m x_l^m + \dots + b_1 x_l + b_0$, where $a_i, b_i \in \mathbb{F}[x_1, \dots, x_{l-1}, x_{l+1}, \dots, x_k]$. The resultant function taken with respect to x_l is defined as the determinant of Sylvester's matrix [9][10]:

$$S_{x_l}(a, b) = \begin{bmatrix} a_n & a_{n-1} & \dots & a_0 & 0 & \dots & 0 \\ 0 & a_n & \dots & a_1 & a_0 & 0 & \dots \\ \vdots & \dots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_n & a_{n-1} & \dots & a_0 \\ b_m & b_{m-1} & \dots & b_0 & 0 & \dots & 0 \\ 0 & b_m & \dots & b_1 & b_0 & 0 & \dots \\ \vdots & \dots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & b_m & b_{m-1} & \dots & b_0 \end{bmatrix}$$

The resultant turns out to be of interest because computing the determinant of the Sylvester's matrix is equivalent to the following :

$$Res_{x_l}(a, b) = \det(S_{x_l}(a, b)) = a_n^m b_m^n \prod_{\substack{(r_a, r_b): \\ a(r_a) = b(r_b) = 0}} (r_a - r_b)$$

For monic polynomials, this is the product of the pairwise difference in the roots $r_a, r_b \in \mathbb{F}[x_1, \dots, x_{l-1}, x_{l+1}, \dots, x_k]$ of each polynomial. This becomes especially important for handling the roots of polynomials in an algebraic fashion, that is, without actually having to explicitly compute their value. Thus, in computation, the roots are never solved for, because that is a problem in and of itself and defeats the purpose of handling the roots algebraically.

Although more exist, four algorithms are generally used to compute the resultant: Sylvester's matrix, Bezout's matrix [11], Collin's modular resultant [12], and Brown's subresultant pseudo-remainder sequences [13]. All four of these algorithms are available in Mathematica and Maple.

The computation of the resultant via Bezout's matrix is quite similar to doing so with Sylvester's matrix, given above. Bezout's matrix is defined as $B(a, b) = (b_{ij})_{i,j=1, \dots, \max[n,m]}$ with elements $b_{ij} = \sum_{k=1}^{\min[i, n+1-j]} a_{j+k-1} b_{i-k} - a_{i-k} b_{j+k-1}$. The determinant of this matrix is a multiple of the resultant of a and b .

Collin's modular resultant algorithm derives from the idea that taking the determinant of a matrix, such as Sylvester's matrix, only requires addition and multiplication operations. Thus, it makes sense to take advantage a homomorphism $\phi : X \rightarrow Y$, where it holds $\phi(x_1 + x_2) = \phi(x_1) + \phi(x_2)$ and $\phi(x_1 x_2) = \phi(x_1) \phi(x_2)$ for any $x_1, x_2 \in X$. In this case, it also holds then for a matrix $Q = (q_{ij})$ with elements $q_{ij} \in X$ that $\phi(\det(Q)) = \det(\phi(Q))$. Collin originally applied this result to the domain of integers \mathbb{Z} using the mapping $\phi : \mathbb{Z} \rightarrow \mathbb{Z}_p$, where p is a prime integer, to take

advantage of the fact that if $|\det(Q)| < \frac{p}{2}$, then $\phi(\det(Q)) = \det(Q)$, and thus, by the previous homomorphic equality, $\det(\phi(Q)) = \det(Q)$. Typically, to make use of smaller primes, several primes and homomorphic mappings are used to do the reduced calculations, the results of which can then be recombined using the Chinese remainder theorem.

Brown's subresultant algorithm is a generalization of the Euclidean algorithm [14] for computing the greatest common divisor of two integers or polynomials. For polynomials $a, b \in \mathbb{F}[x_1, x_2, \dots, x_k]$, assume, without loss of generality, that $\deg(a) \geq \deg(b)$. When the division algorithm holds, a can be written as $a = b \cdot q + r$, where $q, r \in \mathbb{F}[x_1, x_2, \dots, x_k]$ and are called the quotient and remainder, respectively. If the division algorithm does not hold, pseudo-remainders are used instead by introducing a constant multiplier to a . The Euclidean algorithm makes use of the fact that $\gcd(a, b) = \gcd(b, r)$, so we can recursively apply this reduction until the remainder is 0. The subresultant algorithm uses a similar equality but applied to subresultants, which come from submatrices of Sylvester's matrix. These are denoted $Res_{x_l}^j(a, b)$ as the j -th order subresultant, which effectively eliminates j rows and columns from Sylvester's matrix. The following equality then holds:

$$Res_{x_l}^j(a, b) = (-1)^{(n-j)(m-j)} b_m^{n-\deg(r)} Res_{x_l}^j(b, r)$$

This allows us to iteratively simplify the problem, just like the Euclidean algorithm.

B. Meta-algorithms

Mathematica's meta-algorithm to select among the four available algorithms is hidden, but Maple's is available in the documentation on the resultant superfunction and by using the *showstat* command to view the source code. For univariate and bivariate polynomials with rational coefficients (including integral coefficients, even though the two are stored in memory differently), Maple uses modular methods for high degree polynomials, whereas the subresultant algorithm is used for those with lower degrees. In all other cases, Bezout's matrix is used. It's worth noting that, even though Maple offers Sylvester's matrix as an option, it is never considered by the meta-algorithm. Without a doubt, this is because, in most cases, taking the determinant of a smaller matrix, such as Bezout's matrix, is faster than taking the determinant of a larger matrix, such as Sylvester's matrix. However, this assumption ignores important factors, such as the computation of the elements in Bezout's matrix, whether any elements are zero, whether floating point numbers are being used, the kernel of the program, and so on. These factors can regularly make Sylvester's matrix a more viable method; in fact, for randomly sampled polynomials, the proportion of times Sylvester's matrix outperforms all other algorithms is close to the same proportion of times Bezout's matrix does so too.

C. Machine Learning Formulations

Machine learning can be applied to the algorithm selection problem by using classification to categorize a pair of input polynomials into which algorithm will work for best for them. This classification relies only on a set of attributes or features that are taken from a quick analysis of the two polynomials a and b passed in as arguments during the function call $resultant(a, b, x_l)$, the resultant of a and b with respect to the variable x_l . Feature engineering can be completed by analyzing the source code for implemented algorithms or by having a general understanding of each algorithm and then determining the fundamental features of the inputs on which the runtime complexity depends.

The basic features on which to classify are generally easy to spot. For example, it is clear to see that all four available algorithms to compute the resultant depend on the degrees of the input polynomials, although in different manners; the dimensions of Bezout's and Sylvester's matrices directly depend on the degrees of the polynomials, and the coefficient bounds of Collin's modular method changes with the degrees of the polynomials, as well as the number of iterations that occur in the subresultant algorithm.

It is usually the case that algorithms scale differently with regards to changes in any given feature. For example, the cost to take the determinant of Sylvester's matrix scales polynomially with the degrees of the inputs, whereas the coefficient bound in the modular resultant algorithm scales factorially. It's also possible that an algorithm's runtime may change with a given feature, whereas another algorithm's may not change at all, as might happen with the fact that Sylvester's matrix could care less about the polynomial ring, whereas the modular method's runtime may change significantly.

After enumerating many of these features, the list was narrowed down using the ReliefF algorithm [15] to 18 attributes from an initial size of 30. The ReliefF algorithm iteratively takes a random feature vector from the training data and creates a weight for each feature by analyzing the k nearest-hits, which are the closest feature vectors under the L1 norm with the same classification, and the k nearest-misses from each different class, which are the closest feature vectors with a different classification. The full list of attributes used for the resultant case study, along with their descriptions, is given in Table I.

As for labeling, there were four target classes for each of the four available algorithms. Classification into one of these categories means the resultant algorithm corresponding to that class gives the best performance in terms of runtime.

The dataset consisted of 18,346 randomly generated polynomials, which were randomly paired into 9,173 inputs, as the resultant function takes two polynomials as a single input. Since the learning was supervised, the output class for each input in the dataset was obtained by running each of the four algorithms, which are already implemented in Maple and Mathematica, and selecting the resultant algorithm that gave the least runtime over an average of thirty runs.

Neural networks, random forests, k-nearest neighbors, and SVMs with linear and RBF kernels were trained on the given data. The neural network was built with Matlab's pattern recognition tool and consisted of a single hidden layer with 10 sigmoid hidden neurons and a softmax output layer. These formed a feed-forward network that was trained with scaled conjugate gradient backpropagation. The data for the neural network was split into 70% training, 15% validation, and 15% testing.

The random forest model was built based off the model in [16] with 50 random trees that, at each node, split on a random selection of five features. Forests with more trees did not offer significant gains in accuracy compared to the associated cost in runtime, and splitting among less than five features caused a loss in accuracy. For the k-nearest neighbors model, classification was done based solely on the nearest neighbor, since using multiple neighbors caused the accuracy to drop off steeply. A linear search with the Euclidean distance norm was used to find the nearest neighbor. Both the random forest and the k-nearest neighbors model were trained with 10-fold cross validation. Lastly, the SVM model was trained with both a linear and a RBF kernel through libsvm's interface. For these three methods, data was divided into 80% training and 20% testing.

IV. THE SHORTEST TOUR CASE STUDY

The second case study chosen to evaluate machine learning's capabilities as an automatic algorithm selection tool was the shortest tour superfunction, which chooses amongst several algorithms to solve the traveling salesman problem.

A. The Traveling Salesman Problem

Given an undirected, weighted graph G , the traveling salesman problem (TSP) asks for the route with the least weight that visits each vertex once and returns to the starting vertex. G is often taken to be a simple graph, since any parallel edges can be eliminated to the one with the least weight and loops are never used in the final tour. Often, the problem may take a set of points as an input, with the goal now attempting to find the shortest route that visits each point given that the distance between any two points is determined by some metric, such as the Euclidean norm or the Manhattan distance. This is just the original problem applied to a complete graph, where any two points are connected with edge weights determined by their physical distance from each other.

Mathematica provides implementations of the following algorithms to solve the traveling salesman problem for graph objects, which are available through its *FindShortestTour* superfunction:

- Greedy [17]
- Greedy Cycle [17]
- Integer Linear Programming (ILP) [18]
- Simulated Annealing [19]
- Or-Opt [20]
- Two-Opt [20]

TABLE I
 POLYNOMIAL FEATURES

No.	Feature	Description
1.	True/False: $a, b \in \mathbb{Z}[x_1, \dots, x_k]$	Whether or not polynomials have all integer coefficients
2.	True/False: $a, b \in \mathbb{Q}[x_1, \dots, x_k]$	Whether or not polynomials have all rational coefficients
3.	True/False: a or b has floating point coefficients	Whether or not computations will require floating point precision
4.	Number of indeterminants	Number of variables in both polynomials combined
5.	$deg(a)$	Degree of a with respect to x_l
6.	$deg(b)$	Degree of b with respect to x_l
7.	Number of terms in a	Self explanatory
8.	Number of terms in b	Self explanatory
9.	Sparsity rating of a	(Number of nonzero coefficients of a)/($deg(a) + 1$)
10.	Sparsity rating of b	(Number of nonzero coefficients of b)/($deg(b) + 1$)
11.	Number of algebraic coefficients in a	The number of terms that consist of a variable besides x_l
12.	Number of algebraic coefficients in b	The number of terms that consist of a variable besides x_l
13.	Proportion of algebraic coefficients in a	What portion of coefficients are purely algebraic in a
14.	Proportion of algebraic coefficients in b	What portion of coefficients are purely algebraic in b
15.	$lcoeff(a)$	The numeric coefficient of the highest order term in a
16.	$lcoeff(b)$	The numeric coefficient of the highest order term in b
17.	Numeric coefficient with smallest magnitude out of a and b	Self explanatory
18.	Numeric coefficient with largest magnitude out of a and b	Self explanatory

More algorithms are available that are specialized for sets of points instead of graph objects. However, the results in this paper only focus on these six algorithms; that is, they only focus on graph inputs. The same machine learning approach can be applied to classify sets of points into which algorithm is most appropriate for them.

Mathematica's documentation does not make clear which greedy heuristics it employs in its greedy and greedy cycle algorithms. However, it is likely the greedy approach is simplest the nearest neighbors approach that, at each vertex, selects the nearest neighbor as the next point in the tour to proceed to. The greedy cycle algorithm is likely an insertion algorithm that starts with tour consisting of a randomly selected vertex and its nearest neighbor and then iteratively inserting a new vertex k into the tour between two adjacent nodes i, j that already exist in the tour such that the cost of the sum of edges $\bar{i}k$ and $\bar{k}j$ are minimized.

ILP formulates the TSP problem as a minimization problem. [?] describes the problem as follows: For each edge $e \in E(G)$, we define x_e as taking the value 1 if e is part of the tour and 0 otherwise. In addition, $c_e \in \mathbb{R}$ defines the cost of e . Letting $\delta(S) = \{\bar{i}j \in E(G) \mid i \in S, j \notin S\}$ for any set $S \subset E(G)$, we have the following optimization to perform:

$$\begin{aligned}
 &\text{Minimize } \sum_{e \in E(G)} c_e x_e \\
 &\text{s.t. } \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
 &\quad \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, 2 \leq |S| \leq |V| - 1
 \end{aligned}$$

Simulated annealing also treats the TSP problem as a minimization problem. It begins with a random tour through all vertices and then moves on to selecting a new tour from the neighbors of the existing tour. If the new tour is better than the old tour, it is chosen as the preferred tour and the process continues. If it is not better, then it is accepted as the

new tour with a probability dependent on both the difference between tour lengths of the new and old tours as well as the temperature or energy of the annealing process; higher temperatures correspond to a higher probability of acceptance. This temperature is then lowered a bit and the process repeated until a minimum is reached. This process can often result in local minimums rather than global minimums.

Or-opt and Two-opt are both heuristics that begin with an attempted solution, a Hamiltonian cycle that may or may not be the least cost tour. Or-opt is a chain exchange method that repeatedly moves chains of three consecutive vertices to different locations until such movements can no longer provide cost improvements. This then continues with two vertices and then a single vertex. Two-opt is an edge exchange method that iteratively removes two edges from the initial tour and then attempts to reconnect the two remaining chains in a way that lowers the total cost from the previous tour.

B. Meta-algorithm

Like with the resultant superfunction, Mathematica's meta-algorithm implementation is not described in the documentation and is not available as source code. However, it is known that the meta-algorithm guarantees the shortest tour, not an approximation of the shortest tour. However, it is not likely that Mathematica simply defaults to using integer linear programming. The disparity between the time it takes Mathematica's meta-algorithm to compute the shortest tour and the time it takes a direct call to integer linear programming varies too widely for such a statement to be true. What is more likely is that Mathematica makes use of running several algorithms as an attempt to find a solution or opts to run an approximation algorithm in a situation where it is guaranteed it will find the shortest tour.

C. Machine Learning Formulations

As previously mentioned, the greedy, greedy cycle, Or-Opt, Two-Opt, and simulated annealing algorithms all approximate solutions that come with the benefit of a faster runtime than

integer linear programming, which gives an exact solution. However, it is possible for some of the methods to fail to find any solution to the problem at all while the others succeed. Mathematica immediately makes this known to the user.

Since both approximation and exact algorithms are available to choose from in Mathematica, it would not make sense to consider only runtime or approximation error as the sole performance measure on which to base the choice in algorithm. If runtime was the only consideration, the nearest neighbors approach would be the choice every time, just with the cost of large approximation error. Likewise, if approximation error was the only factor of importance, then ILP would be the best choice with zero approximation error but a hefty runtime, assuming it is able to generate a solution. Therefore, it is important to consider both of these factors when deciding what algorithm should be deemed the best algorithm.

To solve this issue, a user specified parameter was used that would allow a user to define how important runtime and approximation error should weigh against each other when the shortest tour superfunction was called. This weight factor, denoted γ , holds a value between 0 and 1, 0 meaning all consideration should be given to minimizing runtime and 1 meaning all consideration should be given to minimizing approximation error. A value of 0.5 would mean both factors should be equally weighted against each other. Machine learning can use this parameter to select an algorithm based on how important runtime and approximation error are to the user, a naive approach to multi-objective classification.

2332 randomly generated graphs, consisting of up to 50 vertices, edge weights between 1 to 100, and all guaranteed of having a Hamiltonian cycle, were used as a training set. Each of the six algorithms were run on each of the 2332 inputs, and their runtimes and approximation errors were recorded. For each algorithm, the percent difference from the best runtime out of all algorithms was noted, as well as the percent difference from the best approximation error out of all algorithms. Then for any algorithm that produced a solution, it's performance would be defined by $\gamma \times (\% \text{ Diff. Approx. Err.}) + (1-\gamma) \times (\% \text{ Diff. Runtime})$. The optimal algorithm for a given weight factor is the one that has the smallest value of this measure.

For different values of γ , the optimal algorithm often changes. To train machine learning to learn this pattern, the 2332 randomly generated graphs were each analyzed with values of γ in the set $\{0, 0.1, 0.2, \dots, 1\}$. In this sense, γ forms a feature of the input that is used for classification because the same graph with different values of γ leads to different classifications. This process ultimately creates a dataset of 25,652 examples on which to train and test.

In addition to the weight factor γ , the features used in classification are described in Table II. In total, there were 21 features to describe a given input. It is highly likely some of the features are repetitive (have information that are given by other features) or are not necessary, as this list of features has yet to be narrowed down by a feature selection process.

Like the resultant superfunction, neural networks, random

forests, k-nearest neighbors, and linear and RBF SVMs were all trained on the given data. All setups for these machine learning models were exactly the same as for the resultant case study, except the neural network had 50 hidden neurons instead of 10 for increased performance.

V. EXPERIMENTAL RESULTS

To train and test the machine learning models, all computation was done on a build with an Xeon E5-2420 CPU, Matrox G200eR2 video controller, and 16 GB of RAM, running 64 bit CentOS 6.8 with an installation of Maple 2015.1 and Mathematica 10.0.2.

A. The Resultant

1) *Setup*: The performance of machine learning as an automatic algorithm selection tool for the resultant superfunction was compared to Maple's and Mathematica's meta-algorithms. More tools for comparison exist, such as Sage, but these tend to default to simply computing the determinant of Sylvester's matrix instead of choosing amongst the different algorithms available. To test the benefits of machine learning, neural networks, random forests, k-nearest neighbors, and SVMs were each trained, and the two most accurate of which, namely neural networks and random forests, were used to replace Maple's and Mathematica's meta-algorithm to select the ultimate resultant algorithm used. These two were then run and timed over several thousand problem inputs. Note, however, that the output class for a pair of inputs was obtained by running all algorithms implemented by the given program and taking the one that gave the least runtime. Since Maple and Mathematica have different kernels which are optimized for different operations, the best algorithm when run under Maple may not be the same as the best algorithm when run under Mathematica for a given input. Thus, it would not be wise to use the machine learning model trained on the best algorithms determined by Maple's runtimes as a replacement for Mathematica's meta-algorithm. Keeping with this paradigm, two datasets were generated, one of which based its output classes on the best algorithm when run under Maple and the other when run under Mathematica.

2) *Accuracy*: The accuracies, or proportions of the time when the algorithm with the least runtime out of all four available resultant algorithms was correctly chosen, for each machine learning approach used are given in Table III. For training on the data generated under the best algorithm choices determined by runtimes in Maple, neural networks outperformed the other machine learning models, correctly selecting the best algorithm 86.63% of the time on the testing data (86.17% and 85.54% for training and validation respectively). Random forests managed to attain 80.71% accuracy, but k-nearest neighbors, RBF kernel SVMs, and linear kernel SVMs all lagged behind at 77.46%, 76.68%, and 76.63% accuracy, respectively.

However, these accuracies indicate that the associated machine learning models are vast improvements over the use of meta-algorithms. Maple's meta-algorithm selects the best

TABLE II
 GRAPH FEATURES

No.	Feature	Description
1.	γ	Weight factor - how important runtime or approximation error is for selecting an algorithm
2.	$ V(G) $	Number of vertices
3.	$ E(G) $	Number of edges
4.	$Max\{cost(e) \mid e \in E(G)\}$	Maximum edge weight in G
5.	$Min\{cost(e) \mid e \in E(G)\}$	Minimum edge weight in G
6.	$ E(G) / \binom{ V(G) }{2}$	The proportion of edges used compared to the complete graph on $ V(G) $ vertices
7.	$\sum_{e \in E(G)} cost(e)$	Sum of weights in G
8.	$Average\{cost(e) \mid e \in E(G)\}$	Average edge weight
9.	$StandardDeviation\{cost(e) \mid e \in E(G)\}$	Standard Deviation of edge weights
10.	$Median\{cost(e) \mid e \in E(G)\}$	Median edge weight
11.	$(\sum_{e \in E(G)} cost(e)) / (100 \cdot \binom{ V(G) }{2})$	Density of edges, where 100 is the maximum possible edge weight
12.	$Max\{deg(v) \mid v \in V(G)\}$	Maximum degree in G
13.	$Min\{deg(v) \mid v \in V(G)\}$	Minimum degree in G
14.	$Average\{deg(v) \mid v \in V(G)\}$	Average degree of a vertex
15.	$StandardDeviation\{deg(v) \mid v \in V(G)\}$	Standard deviation of the degrees of vertices in G
16.	$Median\{deg(v) \mid v \in V(G)\}$	Median vertex degree
17.	$Max\{\sum_{e \in N(v)} cost(e) \mid v \in V(G)\}$	Maximum sum of costs of incident edges for any vertex
18.	$Min\{\sum_{e \in N(v)} cost(e) \mid v \in V(G)\}$	Minimum sum of costs of incident edges for any vertex
19.	$Average\{\sum_{e \in N(v)} cost(e) \mid v \in V(G)\}$	Average sum of costs of incident edges for any vertex
20.	$StandardDeviation\{\sum_{e \in N(v)} cost(e) \mid v \in V(G)\}$	Standard deviation of costs of incident edges for any vertex
21.	$Median\{\sum_{e \in N(v)} cost(e) \mid v \in V(G)\}$	Median of costs of incident edges for any vertex

 TABLE III
 ACCURACIES FOR VARIOUS MACHINE LEARNING MODELS FOR THE
 RESULTANT

Model	Accuracy on Maple data	Accuracy on Mathematica data
Neural Networks	86.63%	78.24%
Random Forests	80.71%	75.97%
KNN	77.46%	69.03%
RBF SVM	76.68%	70%
Linear SVM	76.63%	67%

algorithm out of all four available only 58% of the time. (The accuracy of the meta-algorithm can be determined by looking at the source code or the *userinfo* output to see when a specific algorithm has been called.) However, as previously discussed, considering the meta-algorithm only ever opts for three out of the four available (namely, it ignores Sylvester’s matrix), the accuracy in choosing the best out of three is 72%. Unfortunately, in close to 15% of cases, Sylvester’s matrix proves to be the best choice, so although Maple can select amongst the modular, subresultant, and Bezout algorithms with a 72% accuracy, this choice is still not the best 15% of the time. Nevertheless, in either accuracy measurement, all tested machine learning models outperformed Maple’s meta-algorithm.

Machine learning’s performance dropped significantly when tested against the data generated under Mathematica. Neural networks only attained a 78.24% accuracy, and random forests came close with a 75.97% accuracy. K-nearest neighbors, RBF SVMs, and linear SVMs all underperformed at 69.03%, 70%, and 67% accuracy respectively. Unfortunately, Mathematica hides the implementation of its meta-algorithm and provides no way to see which algorithm was ultimately selected by its meta-algorithm, so the accuracy of Mathematica’s choices

remain unknown.

3) *Time Speedup*: Although there are obvious differences in accuracies between machine learning models and meta-algorithms, these differences are compounded by significantly faster runtimes when machine learning models are used to replace meta-algorithms during the selection of algorithms stage. When applied to a random sample of several thousand inputs, Maple was able to compute the resultant of all inputs in 37,783 seconds with its original meta-algorithm, whereas using the neural network as the selection tool when the resultant superfunction was called yielded a total runtime of only 12,097 seconds, a 68% decrease in runtime. Similarly, in Mathematica, the neural network brought about a 49% decrease in runtime. Random forests, on the other hand, only brought a 46% decrease in runtime to the same sample in Maple and a 37% decrease in Mathematica. Since neural networks have such a better runtime improvement compared to random forests, even though their accuracies only differ by close to 6%, it appears as though when the neural network made an incorrect algorithm choice, the decision it did make was not as bad as when the same situation occurred for random forests. In the case of random forests, the incorrectly chosen algorithm was typically also not the second best algorithm choice. Nevertheless, the high accuracies still resulted in significant performance gains. Since running these tests takes large amounts of server time and resources, the runtime results are limited to just neural networks and random forests and are summarized in Table IV.

It serves to note that there is evidence that Mathematica’s meta-algorithm uses pre-processing to speed up the computation of the ultimately selected algorithms, whereas a direct call to a specific resultant algorithm does not use this pre-processing. This claim comes from the fact that, in occasional instances in the dataset, a call to Mathematica’s

TABLE IV
 RUNTIME IMPROVEMENTS USING MACHINE LEARNING IN PLACE OF
 META-ALGORITHMS FOR THE RESULTANT

Model	Runtime Improvement in Maple	Runtime Improvement in Mathematica
Neural Networks	68%	49%
Random Forests	46%	37%

meta-algorithm yields a faster runtime than single-handedly calling any one of the four algorithms directly. Yet, despite this disadvantage, using machine learning as an automatic algorithm selection tool still manages to outperform Mathematica’s meta-algorithm by a significant factor.

B. Shortest Tour

1) *Setup*: Mathematica was used as a tool for comparison against the machine learning models trained on the TSP problem. For each input and each value of $\gamma \in \{0, 0.1, 0.2, \dots, 1\}$, the timings of each of the six algorithms available to compute the shortest tour of a graph were recorded as well as the length of the tour the generated. Each algorithm’s performance was gauged with the previously discussed performance measure $\gamma \times (\% \text{ Diff. Approx. Err.}) + (1-\gamma) \times (\% \text{ Diff. Runtime})$, where the percent differences are measured with respect to the best approximation errors and runtimes provided out of all the algorithms. The algorithm with the least value given by this measure was deemed the optimal algorithm for the given graph input and value of γ . This process resulted in 25,652 examples to train and test against.

For the value of $\gamma = 1$, there were often multiple algorithms that have the same performance measure. That is, in this particular case, they all returned the shortest possible tour, as runtime is not considered when $\gamma = 1$. When this happened, ties were broken according in the following order, from the highest preferences to lowest: Greedy, Greedy Cycle, Two-Opt, Integer Linear Programming, Or-Opt, Simulated Annealing. The reason for this is that the former algorithms give better runtimes compared to the latter algorithms.

2) *Accuracy*: The proportions of times different machine learning models were able to select the optimal algorithm based on the features of the input graph and the value of γ are given Table V. All machine learning models except for linear SVMs demonstrated exceedingly high accuracies. Random forests and k-nearest neighbors reached 99.6% and 99.62% accuracies, and neural networks and RBF SVMs lagged slightly behind at 96.81% and 95.61% accuracies, respectively. A linear kernel for SVMs, on the other hand, did not sufficiently separate the data and resulted in an accuracy of 38.3%.

3) *Time Speedups*: To test how well machine learning fares at automatic algorithm selection, the total runtimes and path lengths were analyzed from a random sample of 1500 graphs using the algorithm choices decided upon by neural networks, random forests, and Mathematica’s meta-algorithm. These runtimes are given in Table VI. For neural networks and random forests, the percentage by which they improve

TABLE V
 ACCURACIES FOR VARIOUS MACHINE LEARNING MODELS FOR THE
 SHORTEST TOUR

Model	Accuracy
Neural Networks	96.81%
Random Forests	99.6%
KNN	99.62%
RBF SVM	95.61%
Linear SVM	38.3%

upon the meta-algorithm’s runtime is also given. The sum of the all the tour lengths generated by the choice of algorithms from each tool is recorded in Table VII. Note that shortest possible sum of tour lengths is 73,076, which Mathematica attains in all possible cases at the cost of a higher runtime. Since Mathematica’s meta-algorithm has no dependence on γ , the runtimes and sum of tour lengths remain constant for each value of γ .

As expected, as the value of γ increases, the runtimes using machine learning increase while the approximation error decreases. A clear benefit from using a weight factor is that it allows the user to control which is more important in the computation: the runtime, the approximation error, or a combination of both. However, the benefits of using machine learning as a tool for AAS is best demonstrated in the case where $\gamma = 1$. In this case, no explicit attention is given to runtime; rather, the machine learning models have been trained to solely focus specifically on minimizing approximation error. As seen in Table VII, random forests attain the shortest possible tour length in every case when $\gamma = 1$, corresponding to a zero approximation error. This matches Mathematica’s meta-algorithm’s approximation error; however, because of the way machine learning breaks ties between candidate algorithms, random forests also offer a 75.02% improvement in runtime, as seen in Table VI. Not only is the random forest getting the exact solution like Mathematica’s meta-algorithm does, but it is doing so significantly faster. Whereas Mathematica can only manage to focus on one aspect of performance, machine learning is able to balance both runtime and approximation error and see performance improvements at the same time.

VI. CONCLUSION

The results in this paper support the hypothesis that machine learning provides a better alternative for automatic algorithm selection in computational software. When used as a replacement for the resultant meta-algorithms in Maple and Mathematica, neural networks bring about 68% and 49% runtime improvements, respectively. When used in the context of the traveling salesman problem, random forests can weigh multiple facets of performance, specifically runtime and approximation error, against each other using a user specified parameter in order to choose an appropriate algorithm. Machine learning thus allows algorithm selection tools to take into account multiple objectives, whereas meta-algorithms can typically only focus on one performance aspect. However, even when minimizing approximation error in the traveling

TABLE VI
 RUNTIMES FOR DIFFERENT VALUES OF γ

γ	Runtime Under Mathematica (s)	Runtime Under Neural Networks (s)	Percent Improvement Under Neural Networks	Runtime Under Random Forests (s)	Percent Improvement Under Random Forests
0	1797.67	47.989	97.33%	47.871	97.34%
0.1	1797.67	48.080	97.33%	47.871	97.34%
0.2	1797.67	48.025	97.33%	47.876	97.34%
0.3	1797.67	48.026	97.33%	48.655	97.29%
0.4	1797.67	48.822	97.28%	47.893	97.34%
0.5	1797.67	124.897	93.05%	125.579	93.01%
0.6	1797.67	372.427	79.28%	373.926	79.2%
0.7	1797.67	400.618	77.7%	400.429	77.73%
0.8	1797.67	410.127	77.19%	603.909	66.41%
0.9	1797.67	449.523	75%	448.865	75.03%
1	1797.67	449.739	74.98%	448.991	75.02%

 TABLE VII
 SUM OF TOUR LENGTHS FOR DIFFERENT VALUES OF γ

γ	Sum of Tour Lengths Under Mathematica (s)	Sum of Tour Lengths Under Neural Networks (s)	Percent Error Under Neural Networks	Sum of Tour Lengths Under Random Forests (s)	Percent Error Under Random Forests
0	73076	164127	124.6%	164127	124.6%
0.1	73076	164127	124.6%	164127	124.6%
0.2	73076	164127	124.6%	164351	124.9%
0.3	73076	164127	124.6%	164299	124.83%
0.4	73076	164299	124.83%	163998	124.42%
0.5	73076	150199	105.43%	149776	104.96%
0.6	73076	92635	26.77%	92855	27.07%
0.7	73076	82738	13.22%	82738	13.22%
0.8	73076	81407	11.40%	97766	33.8%
0.9	73076	73439	0.5%	73084	0.01%
1	73076	73451	0.51%	73076	0%

salesman problem is the only focus, machine learning can still output the shortest possible tour (zero approximation error) just like Mathematica's meta-algorithm, but with a 75% decrease in runtime as a bonus side-effect. Even larger decreases in runtime are observed when larger approximation errors are allowed by the user.

The overall methodology developed in this paper emphasizes the fact that the process of using machine learning as a tool for automatic algorithm selection is not only straightforward but highly effective. As an example, since most resultant applications require the computation of the resultant dozens or hundreds of times, the runtime improvement would be quite noticeable. In fact, even for a single pair of input polynomials of which to compute the resultant, it is not uncommon for the possible runtimes to range from several seconds to a couple minutes depending on which algorithm is selected. The same situations appear for traveling salesman problems. The high accuracy achieved by machine learning avoids such excessive runtimes.

What remains in the works is to expand this approach to more case studies and explore better formulated multi-objective machine learning models that would allow the optimization of different facets of performance. This would enable an algorithm selection model that can more effectively select the best algorithm based on not only runtime performance, but

also based on memory constraints, least error, and so on. For instance, evolutionary multi-objective optimization algorithms, such as NSGA-II [21] and SPEA-II [22], may be easily used to perform such computation, although the time required to do so may not be acceptable but needs to be empirically determined. All in all, the results in this paper have built a foothold for further exploration into the benefits of machine learning as a tool for automatic algorithm selection, especially in computational software.

ACKNOWLEDGEMENT

This work was funded under NSF grant 1359275. The authors would also like to thank Nathan Harmon for his valued input on the project.

REFERENCES

- [1] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15, 1976.
- [2] Wolfram algorithmbase. <https://www.wolfram.com/algorithmbase/>.
- [3] R. M. Lukose B. A. Huberman and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997.
- [4] C. P. Gomes and B. Selman. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom Reason*, 24(1-2):67–100, 2000.
- [5] M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47:295–328, 2006.

- [6] E. A. Brewer. *Portable high-performance supercomputing: high-level platform-dependent optimization*. PhD thesis, Massachusetts Institute of Technology, 1994.
- [7] E.A. Brewer. High-level optimization via automated statistical modeling. *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 80–91, 1995.
- [8] C. Soares P. Bradzil and D.C.J. Pinto. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.
- [9] S.R. Czapor K.O. Geddes and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.
- [10] A. G. Akritas. Sylvester’s forgotten form of the resultant. *Fibonacci Quarterly*, pages 325–332, 1993.
- [11] R. N. Goldman E. Chionh, M. Zhang. Fast computation of the bezout and dixon resultant matrices. *Journal of Symbolic Computation*, 33:13–29, 2002.
- [12] G. E. Collins. The calculation of multivariate polynomial resultants. *SYMSAC*, pages 212–222, 1971.
- [13] W. S. Brown. The subresultant prs algorithm. *ACM Transactions on Mathematical Software*, 4(3):237–249, 1978.
- [14] W. S. Brown. On euclid’s algorithm and the computation of polynomial greatest common divisors. *Journal of the Association for Computing Machinery*, 18(4):478–504, 1971.
- [15] M. Robnik-Sikonja I. Kononenko and U. Pompe. Relief for estimation and discretization of attributes in classification, regression, and ilp problems. 1996.
- [16] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [17] S. Goyal. A survey on traveling salesman problem.
- [18] Sas/or(r) 9.22 user’s guide. http://support.sas.com/documentation/cdl/en/ormpug/63352/HTML/default/viewer.htm#ormpug_milpsolver_sect020.htm.
- [19] D. Bookstaber. Simulated annealing for traveling salesman problem. *SAREPORT.nb*.
- [20] S. Deneault G. Babin and G. Laporte. Improvements to the or-opt heuristic for the symmetric traveling salesman problem. *Cahier du GERAD*, 2005.
- [21] S. Agarwal K. Deb, A. Pratab and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- [22] M. Laumanns E. Zitzler and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *Eurogen*, 3242(103), 2001.

Classifying and Localizing Epileptic Brain States Using Structural Features of Neuronal Sugihara Causation Networks

Kamal Kamalaldin^{*}, Rory Lewis[†], Chad Mello[†], Dorottya R. Cserpan[‡], Somogyvari Zoltan[‡], Peter Erdi^{*†}, Zsolt Borhegyi[§]

^{*}Kalamazoo College, MI

[†]University of Colorado Colorado Springs

[‡]Wigner RCP, Budapest

[§]MTA-TKI, MTA-ELTE-NAP B-Opto-Neuropharmacology Group

Abstract—Causality is a topic of philosophical and technical debate in scientific fields relating to multivariate systems. Sugihara Causality is a new model for describing causality. We investigate how this model can be applied to highly dimensional neuronal networks where epilepsy is induced. Different brain states will be classified using a neuroclustering algorithm. The time indices of the clustered brain states will be used to discretize the original EEG signal into different epileptic seizures stages. A causality network will be created for each stage from the discretized EEG signal, and analysis on the network will be conducted to find predictive structural patterns in epileptic seizures.

Index Terms—Epilepsy Localization, Brain State Clustering, Information Flow, Neuroclustering, Sugihara Causality

I. INTRODUCTION

The concepts of abstract correspondence, correlation and interpreting causation has been discussed in philosophical literature at least as early as Berkley’s and Locke’s arguments on human perception [1] [2]. Until now, the debate focused on what constitutes a causative effect and how such an effect might be discerned. From philosophy, the debate has moved to empirical science, where different models of causality have been proposed, none of which has been declared the true standard. A particular causality model, Granger Causality (GC), has been widely used in application in the econometric fields [3], and has been the de facto model when causality is concerned. However, while GC behaves best in linear, stochastic systems, it carries its own limitations. Even with extensions to non-linear systems, GC has generally not been seen capable of inferring causality in deterministic systems where feedback loops and nonlinearity are a defining feature. New models of causality have been introduced to attempt to rectify these limitations. Dynamic Bayesian Networks and, more recently, the Convergent Cross Mapping (CCM) are some such models.

The CCM model relies convergence of distance of nearest neighbors in the shadow manifold of pairs of variables [4]. A shadow manifold of variable ω is an E dimensional reconstruction of E delayed signals of ω . Each of these signals is delayed by a scalar multiple of $E\tau$ such that shadow the manifold of ω , M_ω is described as

$$M_\omega = f(\omega(t), \omega(t - \tau), \omega(t - 2\tau), \dots, \omega(t - (E - 1)\tau))$$

. Applying Takens’ embedding theorem, it can be shown that each shadow manifold of a variable is a projection of the dynamic system’s manifold, M , that preserves the topology of M [5], [6], [7]. For example, in a dynamic system like the Lorenz Attractor where the dynamics of each variable is affected by the other variables in the system, it can be said that each variable subscribes to the overall dynamic of the system. Therefore, the state of one variable could be used to infer the state of another variable if they are dynamically linked.

Using this feature of dynamic systems, the CCM model infers causality from the convergence of prediction of one variable’s state based on another’s as L increases, where L is the length of data points considered in the prediction. This implies that L needs to be sufficiently large to allow an observation of convergence. This convergence is the test used to determine Sugihara Causality, named after its author who describes it as a required but not complete definition of causality [4]. This approach is the first step towards more general and applicable causality models since GC. Since the introduction of CCM, it has been shown to be successfully predictive in biological [8], [9], [4], [10], [11] and cosmological [12] applications while showing weaknesses in others [13].

Extensions to and amalgamations of the CCM model are beginning to surface in literature. Clark *et al.* proposed an extension to CCM that relies on measuring the smoothness of the mapping (also called flow) function ϕ , thereby reducing the L length requirement [14]. Wismller *et al.* proposed a Mutual Connectivity Analysis framework for the “analysis and visualization of non-linear functional connectivity in the human brain from resting state functional MRI” [15] which relies heavily on CCM. Tajima *et al.* use the fundamental idea of state space reconstruction to find two measures. The first is *Complexity* which is the best embedding dimension for a certain signal (embedding dimension at which the cross mapping is saturated). The second is *directionality*, the difference in cross map skill or embeddedness between two a pair of signals. With those two measures, they show that the brain exhibits different complexities during conscious and unconscious states. Here, we explore the application of CCM in estimating the causality between neuronal regions by constructing a network of pairwise causality. We then analyse features of such networks during normal and epileptic seizure periods. We attempt to localize the origin of seizures as well as predict their occurrence by using the properties of causal networks.

II. PROBLEM DEFINITION

Given multiple spiking signals from a set of interconnected neuronal regions, can the CCM model identify structural patterns in causal relationships between those regions? Can the patterns and properties of the causality graph resulting from the CCM model be clustered into brain states that represent different stages of epilepsy in the brain? In other words, can a machine learning algorithm be applied to the causal network properties, the model of which could be able to correctly classify epileptic brain states.

III. METHODS

A. Data Collection

EEG data was collected from an 4x8 endodermal electrode array (31 channels, one channel malfunctioned, Fig. 1, 2). During the experiment epileptic seizures were evoked using 4-aminopyridine and EEG data was recorded from the electrode array. The spiking voltage of each recorded electrode is used as a signal and is referred to as a channel.

B. Data Preprocessing

Kernal Current Source Density (kCSD) method was used on the grid of channels to account for possible electrical interference with the direct measurement [16]. The measured potentials produced by kCSD arise as the linear combination of the transmembrane currents, which is a more direct and localized quantity to measure the neural activity. Therefore current source density distribution was calculated by the kCSD method and used for the analysis.

Afterwards, in order to reduce the data size that is operated on, we lumped channels from the same regions together by averaging them

(Fig. 3). In addition to reducing dimensionality, this process also puts emphasis of causality on functional brain regions instead of a local cluster of neurons.

C. Experiments

Using the preprocessed data, a pairwise analysis of the signals will be carried using CCM. For each pair of regions, two directions of causality will be considered. Since there are 12 regions, $12 \times 11 = 132$ unique causality relationships were analyzed. The causality measures were recorded on sliding windows of time segments. The time segment lengths were chosen through a heuristic that focused on attaining multiple causality measures within a second. Once the causality measures were attained for each time segment, a combination of heuristic and statistical measures were used to analyze the significance of the causality measures (see section III-D).

From the causality measures, a graph was constructed for each time segment (Fig. 8).

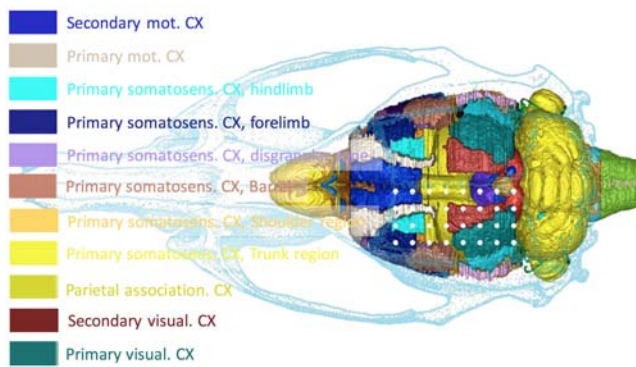


Fig. 1: A composite drawing showing the brain in the skull. The 3D reconstruction of the brain has been made using the maps of the Paxinos atlas, and the localization of cortical areas are indicated by different colors. White points indicate the position of the recording sites of the membrane electrode. Names for the cortical areas are also shown (based on Hjernevik *et al.* and Paxinos *et al.* [17] [18]).

D. Significant Causality Measures

Since every CCM computation returns a real number in the range $[0, 1]$ that represents a relative causality measure, 132 causality measures for each time segment will be returned. Therefore, an important question to consider is how exactly should a causality measure be defined to be significant. Although Sugihara *et al.* [4] and Nes *et al.* [11] carry out a significance test based on altering the signals by random shuffling and Fourier transformation on Phase shift [19], this method has not been implemented on EEG data when applying the CCM model. Therefore, we do not rely on it completely, and take into account several heuristic conditions.

1) *Most Causal Relationship Method*: In this method, a simplistic approach is taken whereby for each region only the highest incoming ρ is considered. While this approach is reductionist by definition, and most likely does not reflect the true causal relationship in the brain, it achieves simplicity in the network, and affords us the possibility of examining which regions could be the most causal in the network. Such regions would have a large out degree which would imply it being a center of causality in the network.

Another important property of this measure is that it could alleviate the problem of downstream causality sensitivity. As shown by Ye *et al.* [20], the CCM algorithm can detect downstream causality, which means that if α causes β and β causes κ , then if $\rho_{\alpha \rightarrow \beta}$

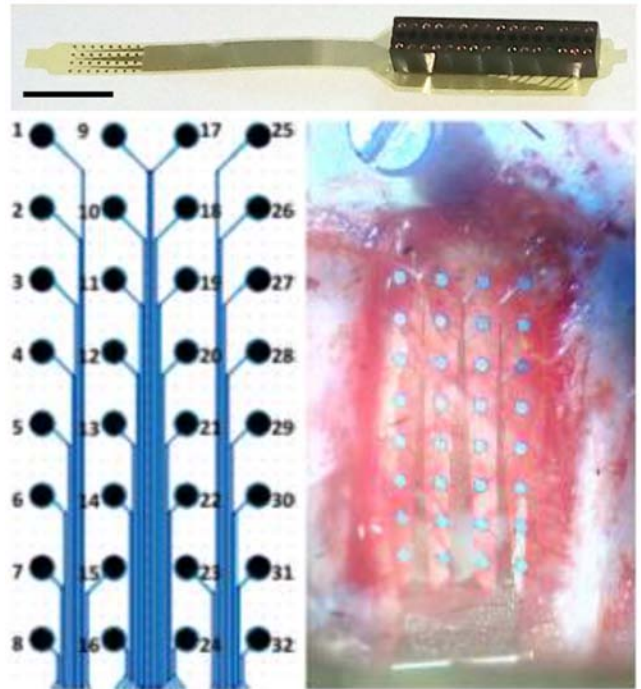


Fig. 2: Photograph of a membrane electrode shows the construction on the top, the numbering (bottom right) and the surgical implantation (bottom left) is also shown. Electrode 1 malfunctioned during recording.

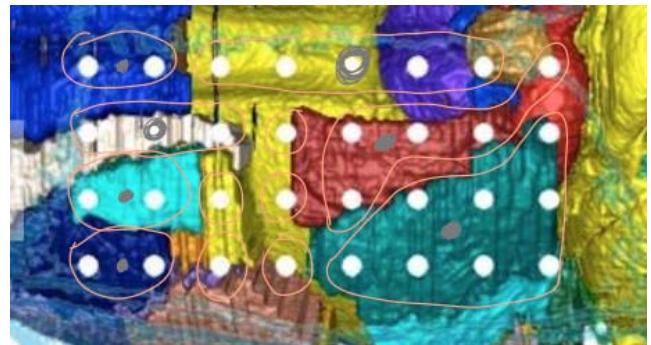


Fig. 3: The distribution and lumping of the brain regions in the brain. A total of 12 region channels were constructed from the initial 31 local channels. The schematic is based on rat brain atlas mapping.

is a significant causality measure from α to β , there can also be a causality measure $\rho_{\alpha \rightarrow \kappa}$ detected to be significant such that the indirect causality is less than the direct causality $\rho_{\alpha \rightarrow \kappa} < \rho_{\alpha \rightarrow \beta}$. This can be an undesired consequence of the model, since we are interested only in the direct relationships, and not necessarily the indirect ones. Indirect relationships could be extrapolated from direct ones. If we assume that all downstream causality measures are evaluated to be less than their direct counterparts, then we expect that taking the most causal relationship would rid of all the indirect relationships that could otherwise be detected, with the expense of also ridding of other direct relationships.

2) *Threshold Method*: In this method, only causality measure over a certain threshold ρ_{th} will be counted as significant. As this is a heuristic measure, the data must first be examined to clarify what is meant by a significant ρ_{th} . With this heuristic comes unavoidable human bias towards refusing generated data. Since the true causality

relationships are not yet uncovered, almost any threshold is certain to be wrong. For example, if the brain was highly connected and regions are highly causal to one another, a neuroscientist who disregards such a possibility would be inclined to choose a high ρ_{th} as to filter many of the causal relationships that could in fact be present. Alternatively, if the brain regions were minimally causal to one another and a neuroscientist disregards that possibility, they would be inclined to choose a low ρ_{th} as to allow for more causal relationships for the model. However biased this method is, if implemented correctly it could provide a list of the most causal relationships to each region, while excluding most indirect relationships.

3) *Fourier Transform and Random shuffling Method*: For a more mathematically grounded significance measure, we also use a bootstrap test where a signal for both channels in a pair is created from the original signal, and the causality measure is significant if it is above a specified α threshold. Another statistical method used to calculate significance is randomizing the data based on bootstrapping the frequency distribution of the signal calculated from the signal's Fourier transform as used by Nes *et al.* [11][7].

IV. RESULTS

By running the CCM algorithm through every pair of channels, we collected 132 causality measures (Fig. 4). To increase the efficiency of our calculations, we also observed the variance of the causality strengths as we decreased the sample size for each calculation (Fig. 5). Most of the causality measures appear to be of high values, suggesting high connectivity between brain regions (Fig. 4). Furthermore, we also notice that many of the pair causality measures appear to be similar within the pair (Fig. 7). For example, when looking at regions 1 and 2 within the first 200 ms, they appear to be equally causal to one another (Fig. 4).

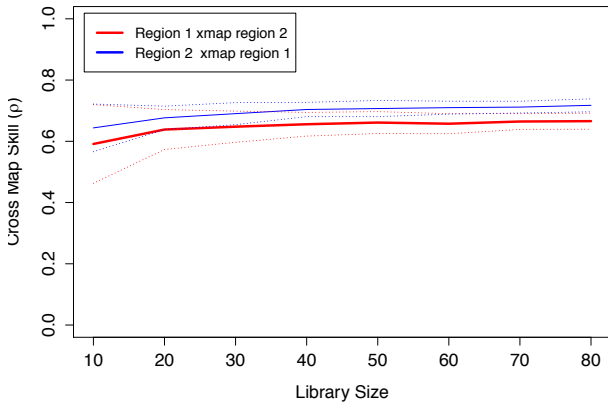


Fig. 4: A sample graph plotting the convergent cross mapping skill (ρ) between region 1 and 2 during the first 200 ms of the experiment. The skill mapping channel 1 to channel 2 is very similar to the one mapping channel 2 to channel 1. This might infer either bidirectional causality or unidirectional forcing. A similar pattern (close ρ value between pairs) was found for most of the pairs. Cross mapping was done with random library samples.

V. DISCUSSION

Initial analysis shows that the causality network is very dense with highly weighted edges. The high density of the graph could have been a side effect of Sugihara's model ability to detect downstream

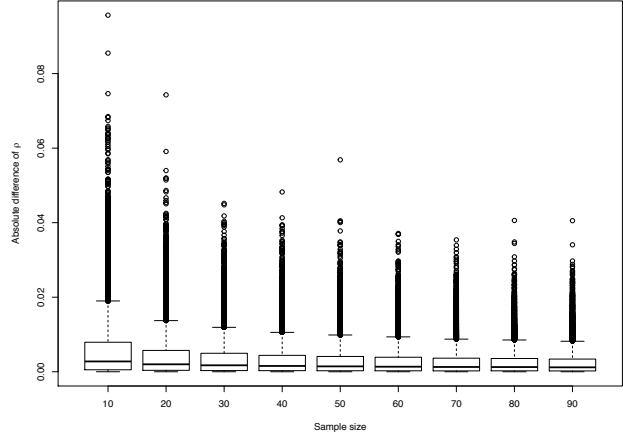


Fig. 5: Difference of rho scores between shown sample size and 100 samples. Decreasing sample size from the default 100 when calculating Sugihara Causality does not have a drastic affect on the acquired result. This shows that the data and method used are robust. Using this analysis, we conduct all further tests on a sample size of 20. Data shown is from the first 5 seconds of the experiment, using a library size of 80. Sliding windows of 200 ms were used, with a sliding step of 50 ms.

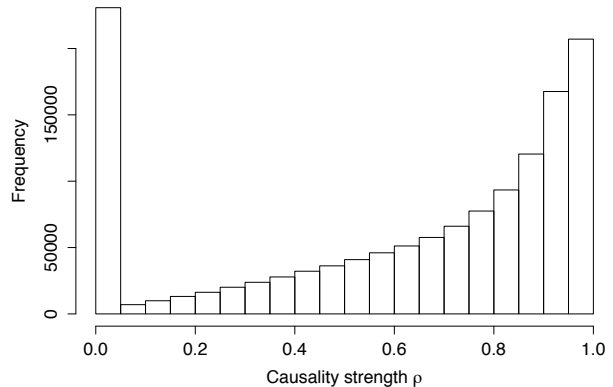


Fig. 6: The distribution of CCM skill (ρ) during the entire experiment. Many relationships appear to be causal in the brain, with equally as many being non-causal throughout the experiment. Causality was calculated from signals of lumped regions after calculating the CSD. Cross mapping was done on every pair of regions with library size of 80, and each pair has two causality directions. Sliding windows of 200 ms were used, with a sliding step of 50 ms.

causality. If that is the case, then many of the causal connections detected are in fact residuals of upstream interactions in the network.

Downstream causality measures can be detected by observing both the magnitude of the cross map skill as well as the time lag that produces the greatest cross map skill (Fig. 3 in [20]). Following the assumption that downstream causality decreases in magnitude as it travels downstream in the network, we can use this to traverse the graph and rid of any paths that decreases in causality as it goes downstream.

Concerning the high possibility of the presence of unidirectional forcing, Ye *et al.* showed unidirectional forcing can be untangled by

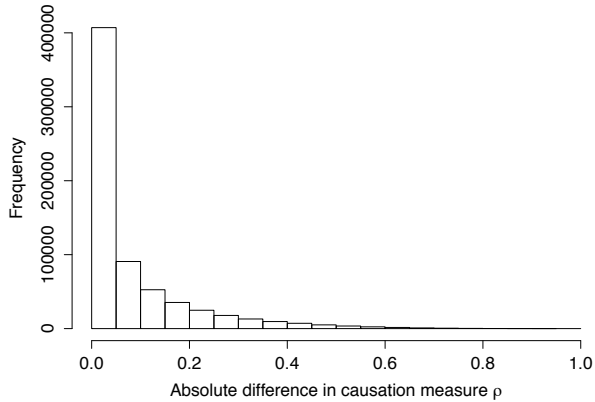
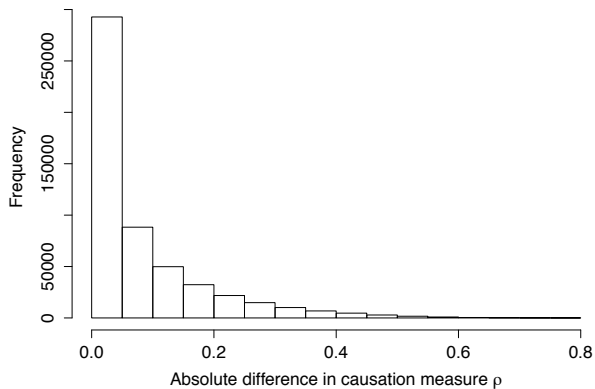

 (a) Absolute difference of ρ between all pairs

 (b) Absolute difference of ρ between pair both of which is at least 0.2

Fig. 7: Difference of causality values within a pair is small, even when accounting for non-existing relationships where both causalities are below 0.2. This similarity between directions of causality in a pair could imply a bidirectional relationship between most regions, or could alternatively imply a unidirectional forcing (synchrony) phenomenon. Sliding windows of 200 ms were used, with a sliding step of 50 ms. A library size of 80 was used.

inspecting the greatest time lag of the two that produces the highest causality measure (Fig. 2 in [20]). In order to allay the problem of unidirectional forcing, the best lag of each pair is considered. This is a tricky problem because there is no clear range for which to test the lag. This is because the time delay for neuronal activity is yet studied, and how that translates to EEG data could be tricky. We reserve the use of this method due to its computational complexity which would add to the already high time complexity of the analysis.

The theoretical implications of this model could present a novel representation of information flow in the brain and determining causality within the brain. If the graph output of this method is reliable, it could help outline information flow within the brain, much like one would observe in a magnetic wave flowing through an fMRI recording.

VI. FUTURE WORK

Future work will focus on verifying this model through controlled experiments. Such experiments could be in the form of stimulating

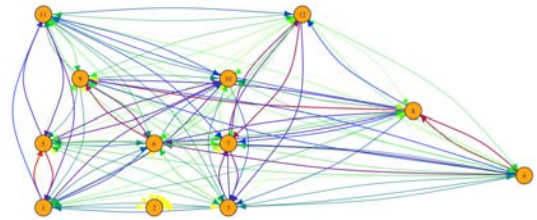


Fig. 8: Sample graph constructed from calculating the Sugihara causality between brain regions through CCM. Lumped brain regions correspond to the gray circular mapping in Fig. 3. Edge colors represent the strength of the causal relationship. From weakest to strongest: Yellow, Green, Blue, Red.

a part of the brain (e.g. shining strong light on an the eye to excite the visual cortex) and observing the model's behavior. One would expect a high value of out degrees from the specific region during such an experiment, as it attempts to convey a considerable portion of information to the rest of the brain. Moreover, a clear proof should be presented as to what the most reliable time window and step size ought to be when producing the causality graphs. Such a task can be done by measuring graph similarity of the same time segment as the time window gets shortened incrementally.

Continuing our efforts, we would like to integrate the neuro-clustering algorithm developed by Lewis and Mello into this work [21]. This would allow us to discretize the EEG data into frames of epileptic seizure stages. Once these stages are identified and compartmentalized, the kCSD method will be applied to account for experimental design errors. A pairwise causality network will then be constructed using the Sugihara causality model, and the origin of the epilepsy will be localized during the initiation of the seizure.

Furthermore, the feature set of the neuroclustering algorithm could be augmented with the information of edge weights of the pairwise causation graph. The effect of such an integration could be tested to see if it improves classification metrics. Similarly, the neuroclustering algorithm could be used to label seizures in neural data which then our pairwise causality algorithm could be tested against to see if the clustering of edge weights clusters epilepsy segments separately.

VII. CONCLUSION

The paper shows promising initial results using the Sugihara CCM model to construct causality graphs between brain regions. We find that the brain network for this experiment is highly causal with a range of time windows. This, however, could be due in part to experimental design limitations, where the electrodes were 1 mm apart which might have caused electrical interference. To to limit this phenomenon we used kCSD to preprocess the data. Initial results show a time varying graph in which information flow can be tracked. At the moment, more analysis is required to make a conclusions on the capabilities of the pairwise causality graph model. Some of most incurring difficulties to overcome are the running complexity of the kCSD preprocessing and CCM algorithm required for the analysis

of the amount of pairs in a large network, and the mathematical representation of information flow within the time-dependent graph. We plan on using the Neuroclustering algorithm to discretize the EEG data into epileptic seizures, extract causal network features from the stages, and train a k-means learning algorithm on the created feature set. The implications of these findings could relate more generally to discoverability of causality in modeling scalable natural phenomena. Real world applications manifest in localization of epilepsy in the brain. Furthermore, if the technique of distinguishing causal networks in systems and clustering their properties is successful, it could be a clear indication that the Sugihara causality model is able to detect causation in extremely complex systems comparable to the human brain..

ACKNOWLEDGMENT

Special thanks goes to Dr. Somogivari and Dorrottya R. from Winger RCP, Budapest for providing the neuronal data used in this paper. Furthermore, Dr. Peter Erdi is thanked for his patient guidance of this research topic which was instrumental for the production of this paper.

REFERENCES

- [1] J. Locke, *An essay concerning human understanding*, 1841.
- [2] G. Berkeley, *A treatise concerning the principles of human knowledge*. Philadelphia: JB Lippincott & Company, 1874.
- [3] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, pp. 424–438, 1969.
- [4] G. Sugihara, R. May, H. Ye, C.-h. Hsieh, E. Deyle, M. Fogarty, and S. Munch, "Detecting causality in complex ecosystems," *science*, vol. 338, no. 6106, pp. 496–500, 2012.
- [5] G. S. Paul A. Dixon, Maria J. Milicich, "Episodic fluctuations in larval supply," *Science*, vol. 283, no. 5407, pp. 1528–1530, 1999.
- [6] E. R. Deyle and G. Sugihara, "Generalized theorems for nonlinear state space reconstruction," *PLoS One*, vol. 6, no. 3, p. e18295, 2011.
- [7] F. Takens, *Detecting strange attractors in turbulence*. Springer, 1981.
- [8] E. R. Deyle, M. Fogarty, C.-h. Hsieh, L. Kaufman, A. D. MacCall, S. B. Munch, C. T. Perretti, H. Ye, and G. Sugihara, "Predicting climate effects on pacific sardine," *Proceedings of the National Academy of Sciences*, vol. 110, no. 16, pp. 6430–6435, 2013.
- [9] X. Wang, S. Piao, P. Ciais, P. Friedlingstein, R. B. Myneni, P. Cox, M. Heimann, J. Miller, S. Peng, T. Wang, H. Yang, and A. Chen, "A two-fold increase of carbon cycle sensitivity to tropical temperature variations," *Nature*, vol. 506, no. 7487, pp. 212–215, 02 2014.
- [10] J. C. McBride, X. Zhao, N. B. Munro, G. A. Jicha, F. A. Schmitt, R. J. Kryscio, C. D. Smith, and Y. Jiang, "Sugihara causality analysis of scalp eeg for detection of early alzheimer's disease," *NeuroImage: Clinical*, vol. 7, pp. 258–265, 2015.
- [11] E. H. van Nes, M. Scheffer, V. Brovkin, T. M. Lenton, H. Ye, E. Deyle, and G. Sugihara, "Causal feedbacks in climate change," *Nature Climate Change*, vol. 5, no. 5, pp. 445–448, 2015.
- [12] A. A. Tsonis, E. R. Deyle, R. M. May, G. Sugihara, K. Swanson, J. D. Verbeten, and G. Wang, "Dynamical evidence for causality between galactic cosmic rays and interannual variation in global temperature," *Proceedings of the National Academy of Sciences*, vol. 112, no. 11, pp. 3253–3256, 2015.
- [13] J. M. McCracken and R. S. Weigel, "Convergent cross-mapping and pairwise asymmetric inference," *Physical Review E*, vol. 90, no. 6, p. 062903, 2014.
- [14] A. T. Clark, H. Ye, F. Isbell, E. R. Deyle, J. Cowles, G. D. Tilman, and G. Sugihara, "Spatial convergent cross mapping to detect causal relationships from short time series," *Ecology*, vol. 96, no. 5, pp. 1174–1181, 2015.
- [15] A. Wismüller, X. Wang, A. M. DSouza, and M. B. Nagarajan, "A framework for exploring non-linear functional connectivity and causality in the human brain: Mutual connectivity analysis (mca) of resting-state functional mri with convergent cross-mapping and non-metric clustering," *arXiv preprint arXiv:1407.3809*, 2014.
- [16] J. Potworowski, W. Jakuczun, S.Ł ski, and D. Wójcik, "Kernel current source density method," *Neural computation*, vol. 24, no. 2, pp. 541–575, 2012.
- [17] T. Hjernevik, T. B. Leergaard, D. Darine, O. Moldestad, A. M. Dale, F. Willoch, and J. G. Bjaalie, "Three-dimensional atlas system for mouse and rat brain imaging data," *Frontiers in neuroinformatics*, vol. 1, p. 4, 2007.
- [18] G. Paxinos, C. Watson, P. Carrive, M. Kirkcaldie, and K. Ashwell, "Chemoarchitectonic atlas of the rat brain," 2009.
- [19] W. Ebisuzaki, "A method to estimate the statistical significance of a correlation when the data are serially correlated," *Journal of Climate*, vol. 10, no. 9, pp. 2147–2153, 1997.
- [20] H. Ye, E. R. Deyle, L. J. Gilarranz, and G. Sugihara, "Distinguishing time-delayed causal interactions using convergent cross mapping," *Scientific reports*, vol. 5, 2015.
- [21] H. Ali, Y. Shi, D. Khazanchi, M. Lees, G. D. van Albada, J. Dongarra, P. M. Sloom, J. Dongarra, R. Lewis, C. A. Mello, and A. M. White, "Proceedings of the international conference on computational science, iccs 2012 tracking epileptogenesis progressions with layered fuzzy k-means and k-medoid clustering," *Procedia Computer Science*, vol. 9, pp. 432 – 438, 2012.

Optimization of Neural Network Architecture for Biomechanic Classification Tasks with Electromyogram Inputs

Alayna Kennedy, *Pennsylvania State University*, Rory Lewis, *University of Colorado at Colorado Springs*

Abstract—Electromyogram signals (EMGs) contain valuable information that can be used in man-machine interfacing between human users and myoelectric prosthetic devices. However, EMG signals are complicated and prove difficult to analyze due to physiological noise and other issues. Computational intelligence and machine learning techniques, such as artificial neural networks (ANNs), serve as powerful tools for analyzing EMG signals and creating optimal myoelectric control schemes for prostheses. This research examines the performance of four different neural network architectures (feedforward, recurrent, counter propagation, and self organizing map) that were tasked with classifying walking speed when given EMG inputs from 14 different leg muscles. Experiments conducted on the data set suggest that self organizing map neural networks are capable of classifying walking speed with greater than 99% accuracy.

Index Terms—Electromyogram, prostheses, neural networks, biomechanical analysis, machine learning, myoelectric control schemes, self organizing maps, pattern recognition algorithm



1 INTRODUCTION

MODERN assistive prosthetic devices commonly use a myoelectric control scheme, which adjusts the function of a prosthetic device given electromyogram (EMG) inputs, and which has been proven an effective method of man-machine interfacing between user and prosthetic [1]-[4]. However, despite the significant development of the prosthetic industry over the past decade, high-accuracy commercial prostheses remain too expensive for the average middle-class amputee to afford [5], [6]. Even the most accurate devices possess multiple issues, including difficulty actuating multiple degrees of freedom and low accuracy without a high number of electrodes [7]. Within academia, computational intelligence pattern recognition techniques of EMG analysis provide accurate results, but are often computationally expensive [8]. Therefore, academic models result in limited real-world improvements in the control of prosthetic devices, since available cheap prostheses do not possess the computation power necessary to run complex pattern recognition algorithms. An ideal prosthetic control scheme would be able to achieve accuracy in classification tasks while remaining simple enough to be implemented in inexpensive devices [6], [8].

Among the common electrophysiological signals, EMG recordings are used most extensively in man-machine interfacing because of their non-invasiveness,

relatively easy application, and richness of neural information [3], [8]. While other biosignals like electroencephalograms (EEGs) and electrooculograms (EOGs) can be used to predict human movement, EMG signals have become the standard biosignal for myoelectric prostheses because they directly transmit electrical signals from the muscle during periods of contraction or relaxation [3], [9], [10]. In addition, EMG signals have been shown to precede muscle kinematics by 100ms; therefore, they can be used to predict human movements to create a prosthetic device that would analyze EMG signals and transmit them to a prosthetic, which would then move as if it were the amputees biological limb [1], [2].

The EMG signal is the sum of the electrical activity of the muscle fibers, as triggered by the impulses of activation of the innervating motor neurons [3], [8], [10]. Surface EMGs are obtained by convolution of each motor neuron spike train by the motor unit action potential (MUAP) and have mathematically the same expression as the neural efferent signal [10].

Equation 1 shows a simple model of the EMG signal:

$$x(n) = \sum_{r=0}^N h(r)e(n-r) + w(n)$$

where $x(n)$ is the modeled EMG signal, $e(n)$ represents the firing impulse, $h(r)$ represents the MUAP, $w(n)$ is the zero mean additive white Gaussian noise, and N is the number of motor unit firings [8].

Despite the rather direct relation between a motion and the expressed EMG signal, however, there remain several open issues before the control of prostheses by EMG will reach the ideal characteristics needed for

- *A. Kennedy is an undergraduate student at The Pennsylvania State University studying Engineering Science and Mechanics. This project is a result of her participation in a Machine Learning REU (Research Experience for Undergraduates) with the Department of Computer Science at University of Colorado, Colorado Springs, CO 80918. E-mail: aakennedy3696@gmail.com*

widespread acceptance by patients [4], [8]. Raw EMG data alone provides very valuable information about human biomechanics in a fairly useless form, since the amount of noise in the data require it to be processed, quantified, cleared of noise, and decomposed into individual MUAPs (Fig. 1.) before it can be used in any significant way [7], [10], [11]. While difficult, the effective analysis of EMG signals can be achieved through wavelet analysis, artificial neural networks (ANNs), and different clustering algorithms [10].

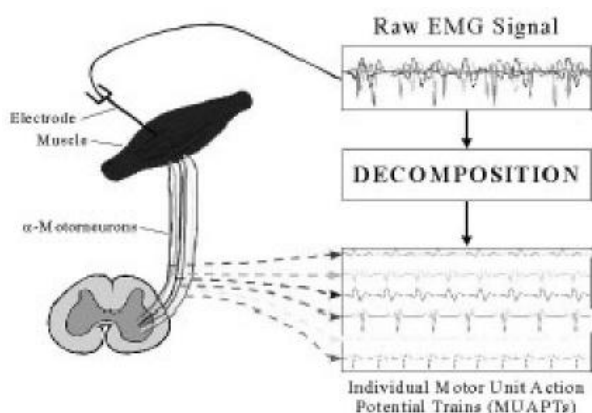


Fig. 1: Decomposition of EMG signals into individual MUAPs

While many studies have been done on the continuous analysis of EMG data using neural networks and complex pattern recognition algorithms, little work has been done on less computationally expensive classification tasks [4]. While regression tasks provide detailed information in the form of bodily spatial coordinates, they also require more electrodes and achieve a lower degree of accuracy than many classification tasks [7]. Although classification tasks only output a discrete category, they can also provide a great deal of information about future movement. For example, walking speed can determine multiple factors about human gait [12]. Other applications of classification tasks to prosthetic technology can be found in Gandolla and Gehani's studies, where trained ANNs classified distinct hand movements, resulting in a hand prosthetic performing the movement [1], [3].

This study aims to find a network architecture that can accurately classify kinematic information from EMG signals while remaining simple enough for application in affordable prostheses, since one of the most significant applications of EMG analysis technology is within the field of orthotics and prosthetics [5].

This study aims to find the best way to use ANNs to classify human movement from EMG data, by measuring the performance of multiple neural network architectures when tasked with classification tasks. The different ANN architectures will attempt to classify walking speed into one of five categories given surface EMG inputs from 14 different leg muscles [13].

2 PREVIOUS WORK

The parallel computation power and nonlinear operations performed by the human brain inspired the original ANNs, so when creating biomimetic control schemes for prosthetics, many researchers turned to these networks to create efficient myoelectric devices that worked with the human body [2]. While some of the earliest attempts to create myoelectric prostheses date back to the 1970s, advanced pattern recognition techniques did not emerge until the 1990s with the rise of more accurate machine learning techniques like neural network back-propagation and time-series analysis of inputs [3], [14]. One of these developments in myoelectric control was a dynamic recurrent neural network which accurately predicted spatial coordinates of arm trajectory [15]. The development of recurrent networks, in which all neurons are interconnected, resulted in a marked improvement in the accuracy of spatial coordinate prediction of the body given EMG inputs. Networks like Drayes and Cheron's also implemented time-series backpropagation, which took into account the previous values of EMG input instead of just the immediate input channel [14], [15].

Other networks modeled EMG inputs with an autoregressive (AR) model, and then passed them through an ANN to control the movements of a virtual prosthetic [3]. The results from these studies have shown that controlling prostheses through an ANN recognition of EMG patterns can optimize the number of electrodes, provide greater degrees of freedom for the device, and accurately predict user intent before muscle movements [1], [7], [20]. Two important network architectures for classification of EMG data are the Kohonen Network, and a cascade architecture network with a preprocessing step involving Kohonen maps.

2.1 Kohonen Map:

Also referred to as a self-organizing map (SOM), this network implements unsupervised learning, which maps points in the input space to points in the output space while preserving the topology [16]. Normally, the input space is of high dimension while the output is usually two dimensional. The network identifies the spatial concentration of the network activity that is best tuned to the present input [3], [16].

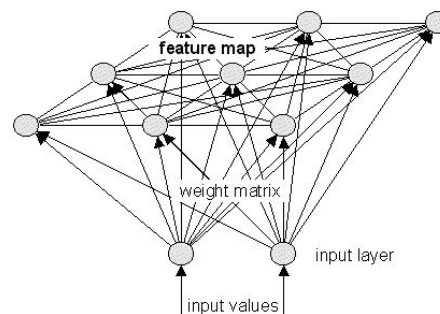


Fig. 2: Detailed view of a Kohonen self organizing map network

The following six steps explain the algorithm for producing Kohonen maps:

Step one: Select network topology: The arrangement of the clusters for the network can be square, circular, etc.

Step Two: Initialize weights to random values: The weight matrix represents the connections between the neurons of the network, and it is randomized to initial values which are subsequently modified during the learning phase

Step Three: Select a pattern: Chooses an input pattern, x , from the input examples.

Step Four: Find best matching unit: The node with the weight vector most similar to the input vector, defined as the node with the smallest Euclidean distance to the input weights, is selected as the matching node.

Step Five: Update weights to all nodes: The winning node and its topological neighborhood are updated by the SOM algorithm according to the equation:

$$m_i(t+1) = \begin{cases} m_i(t) + \alpha(t)[x(t) - m_i(t)] & \text{if } i \in Nc(t) \\ m_i(t) & \text{if } i \notin Nc(t) \end{cases}$$

Where $m_i(t+1)$ is the new weight, and $m_i(t)$ is the old weight, $\alpha(t)$ is the learning rate factor ($0 < \alpha(t) < 1$), and $t = 0, 1, 2, \dots$ is an integer representing the discrete time coordinate.

Step Six: Iteration: Repeat steps 1 to 5 for all input patterns and the repeat for a pre-determined number of iterations [3], [16].

SOMs have been used widely in many applications, including in EMG clustering classification. This network can be used as a preprocessing stage for other ANN architectures such as a Cascade Architecture with Feature Maps (CANFM).

2.2 Cascade Architecture with Feature Maps (CANFMs):

The CANFM network implements a cascaded architecture of neural networks with feature maps (CANFM) [17]. This network first passes the data through an unsupervised Kohonen self-organizing map, outputting 2D coordinates onto the x and y axes of the 2D topological net, which both reduces the input dimensions of the EMG data channels and removes some noisy data from the original inputs [16], [17].

This network first randomizes the initial weights of the network, then passes the values through a self-organizing map (SOM) where the EMG values are clustered. The unsupervised SOM can find a winning neuron on the 2-D topology map to represent the original pattern, and the x and y coordinate of this winning neuron y_c become the input values for a back-propagation neural network (BPNN) [17], [18]. After reduction of the input space using Kohonen's SOM, the three sets of 2-D coordinates (six newly condensed features) are fed into the BPNN for further classification. Huang chose

the BPNN as the post-classifier of CANFM because of its learning ability and fast recall speed. In the case of Huang's study, there are eight postures to be classified, so the BPNN has eight output nodes (Fig. 3) [17].

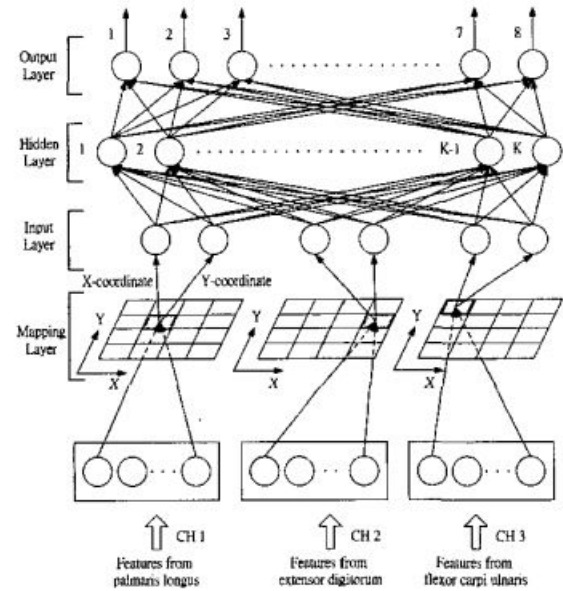


Fig. 3: Detailed information of CANFM

3 ACQUISITION OF EMG DATA

The EMG data for this study was collected by Hof et al and made available to the public via the Clinical Gait Analysis Database [13].

3.1 Collecting Data

Surface EMGs of 14 leg muscles were recorded from two homogeneous groups ($n=9$ and 11 , respectively) of young healthy male subjects (mean age 22 years (S.D. 1.5), stature 1.85 m (S.D. 0.05), leg length 0.98 m (S.D. 0.04), body mass 73 kg S.D. 8). The average personal data of both groups was matched to compensate for the division in the two groups. Subjects walked barefoot on a 10 m indoor walkway at speeds of 0.75, 1.00, 1.25, 1.50, and 1.75 m s^{-1} .

3.2 Pre-processing: Filtering and Blocking

Compared to other biosignals, EMG signals are difficult to analyze due to their small amplitude, which makes them highly subject to both internal and external electrical noise. Internal noise occurs when an electrode picks up signals from more than one MUAP and overlaps their signals, while external noise can result from equipment noise, electromagnetic radiation, or motion artifacts. Therefore, preprocessing and filtering is essential to obtain reliable raw EMG data [3], [10], [13].

In this study, the EMGs were high-pass filtered at 20 Hz, rectified and smoothed with a 25 Hz third order Butterworth low-pass filter. Smoothed rectified EMGs were,

after A/D conversion with a sample frequency of 100 Hz, linearly interpolated to 100 points per stride, triggered by heel contact of the leg of interest. The recorded steps were screened to exclude those with obvious artefacts or incorrect foot contacts. In this way for every individual i , normalized speed v , and muscle m , average individual profiles $\mathbf{e}(\mathbf{p}, \mathbf{m}, \mathbf{v}, \mathbf{i})$ were determined from at least 10 steps over $p=1-100\%$ of the gait cycle [13].

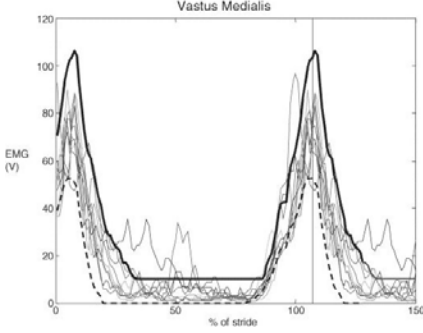


Fig. 4: Average EMG profiles of vastus medialis muscle after normalizing. Also shows low and high limits. Time is given as a percentage of stride, starting at heel contact, but in a scale running 0-100-50%, in order to represent better the activity around heel strike.

4 NEURAL NETWORK CREATION

The EMG data taken from the Hof study was split into training, validation, and testing data in a 2-1-1 ratio, as is considered best practice when training neural networks [19]. The data was then normalized to a range of values between 0 and 1 using the following equation [12].

$$f(x) = \frac{(x - d_L)(n_H - n_L)}{(d_H - d_L)} + n_L$$

where x is the value to be normalized, d represents the high and low values of the data, and n represents the high and low normalization range desired [20].

Four different neural networks architectures were created, a multilayer feedforward perceptron network, a recurrent neural network, a self organizing map (SOM), and a counter propagation neural network (CPN). Each network performed classification tasks on the EMG data, and their performance was recorded. In the first type of architecture, feedforward perceptrons, the networks all had a single hidden layer and were tested with four different combinations of activation functions: linear, radial basis function (RBF), sigmoid (SIG), and hyperbolic tangent function (TANH) [21]. The TANH activation function uses the hyperbolic tangent function

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (1)$$

and has proved a powerful tool for classification tasks [20]. However, in the analysis of EMG data with neural networks, the SIG function

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

is more typically used [15], [19]. The feedforward networks were also tested with varying numbers of hidden layer neurons to ascertain the optimal number for both speed and accuracy of classification. All of the feedforward networks implemented a resilient backpropagation (RPROP) learning algorithm, which is a gradient-based optimization technique similar to the more common regular backpropagation. RPROP is often faster than training with backpropagation and does not require any free parameter values to be specified. RPROP works similarly to traditional backpropagation, except an individual delta value is calculated for each connection. These delta values are gradually changed until the neural network weight matrix converges on a potentially ideal weight matrix [19], [20], [22].

In each iteration of RPROP, the new weights are given by [22]:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$$

The second type of architecture tested was a recurrent neural network (RNN) architecture. RNNs are a subset of ANNs where connections between units form a directed cycle. This allows the internal state of the network to exhibit dynamic temporal behavior, using their internal memory to process sequences of inputs. The two types of RNNs tested were a three-layer Elman network and a Jordan network. In the Elman simple recurrent network, context units connected to the hidden layer maintain a copy of the previous values of the hidden units, allowing the network to maintain a memory of the previous time step [23]. Jordan networks are similar, but the context units are fed from the output layer instead of the hidden layer [24].

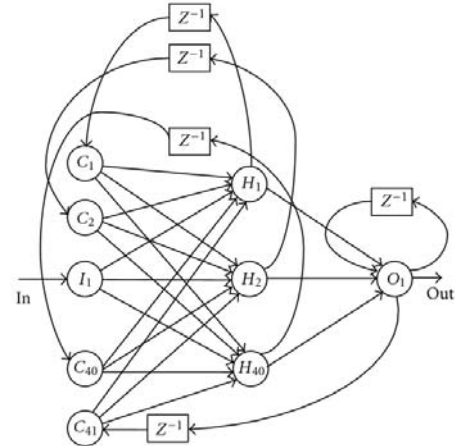


Fig. 5: A structure of the trained Elman-Jordan neural network

The third type of architecture tested were SOM neural networks. This network, unlike the previous two, was trained using unsupervised learning to produce a two-dimensional discretized representation of the input space of the training samples, called a topology map. In addition, the SOM applied competitive learning techniques to improve the network. The SOM architecture was tested

with two different training algorithms: K-means nearest neighbor training and cluster copy training.

The fourth network type, counter propagation networks (CPNs) are similar to CANFM networks, consisting of an outstar network and a competitive filter network. Each neuron in the input layer is processed through a Kohonen network which categorizes the input pattern, serving as the hidden layer for the network [25]. The outputs of the Kohonen map are then filtered through an outstar array which reproduces the correct output pattern for the category. Training is done in two stages; first the hidden layer learns to categorize the patterns and then the weights for that layer become fixed. Then the output layer is trained. One of the advantages of CPN and Kohonen networks is that the training phase requires a relatively small number of epochs, usually several hundred, which is considered a tiny number compared to other ANNs, such as convolutional and deep neural networks which require thousands of iterations of training [3].

For each of the four network architectures chosen, all of the networks were trained, tested, and evaluated for error four times, then the error results over each of the four runs were averaged. Each network was trained using the training and validation, then the method was evaluated using the testing data. When training the network, a maximum error percentage of 1%, max step of 50 and initial update of 0.1 were used [26].

5 RESULTS FROM THE COMPARISON OF NETWORK ARCHITECTURES

Initial results on optimal classification for multiple neural network architectures corroborated results from previous studies, showing that in feedforward networks, the number of neurons in the hidden layer does not significantly affect accuracy of classification, but a higher number of hidden neurons allows the network to train faster. The feedforward networks tested all had 14 inputs, corresponding to the 14 EMG channels, and used a TANH function for both the hidden and output layer activation.

First, the network was tested with 7 hidden layer neurons, half the number of input neurons, which resulted in a 4.782% error in 102 seconds of training time. The number of hidden layers was increased by two during each subsequent experiment, and both the error and training time recorded, with the final test consisting of a network with 30 hidden layer neurons. While error decreased slightly with a greater number of hidden neurons, the more significant change was the decrease in training time with more hidden layer nodes. Both the error and the training time leveled off at around 28 hidden layer neurons, with an error value of 4.4% and a training time of 0.35 seconds. Therefore, we can state generally that the optimal number of hidden neurons to minimize both training time and error for this experiment is about twice

the number of input neurons, a result which has been produced in several previous studies [25].

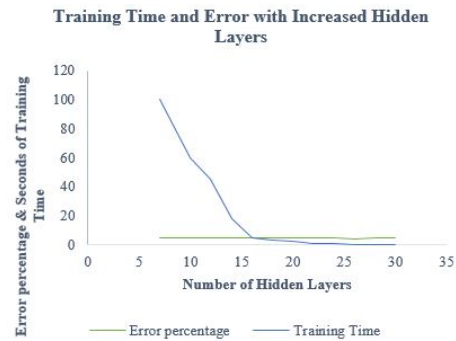


Fig. 6: Graph of training time and percent error of feedforward network vs the number of hidden neurons in the hidden layer of the perceptron.

In analyzing the performance of the feedforward neural network architecture, we can see that using the TANH activation function for both the hidden layer and the output layer activation provides the most accurate results for classification tasks, with an average error of 0.99%. Using the SIG activation function in the hidden layer and output layer activation resulted in an error of 12%, a result which is surprising since most neural networks with EMG inputs use the sigmoid activation function with relative success. However, most studies using the SIG function utilize networks which perform regression tasks to output spatial coordinates, while our network classified the EMG data into 5 discrete walking speeds.

Furthermore, both the feedforward network with TANH activation for the hidden layer and linear outputs and the feedforward network with an RBF activation function performed poorly when classifying EMG data, with average errors of 15%. While the TANH & linear network was included primarily to serve as a control for the TANH & TANH network, the RBF function has been shown to perform well in networks that classify numerical data. However, these results suggest that RBF networks do not perform well on classification tasks with small-amplitude EMG data.

TABLE 1
ERROR PERCENTAGES OF NEURAL NETWORK ARCHITECTURES

Machine Learning Method	Average Accuracy
Feedforward 1: Activation TANH & Linear	15%
Feedforward 2: Activation SIG & SIG	11%
Feedforward 3: Activation TANH & TANH	0.99%
Feedforward 4: RBF Activation	15%
Recurrent 1: Elman Network	13.7%
Recurrent 2: Jordan Network	15.3%
Counter Propagation Network	15.4%
Self Organizing Map 1: K-Means nearest neighbor training	2.1%
Self Organizing Map 2: Cluster copy training	0.27%

Fig. 7: Average error percentages of four feedforward networks with varying activation functions and two SOM with different training

techniques.

The performance of recurrent neural networks on classification tasks was comparable to the TANH and linear feedforward neural network, with the Jordan recurrent network producing an error of 15.3% and the Elman recurrent network producing an error of 13.7%. While dynamic recurrent neural networks have previously performed accurate classifications of EMG data when diagnosing neuromuscular diseases or prehensile human postures, they do not appear to perform well when given normalized EMG inputs to classify walking speed [17][24]. This result could be due to the generalized normalization of the data to values between 0 and 1, which has been shown to improve results in feedforward networks, but is often not used with recurrent systems [23].

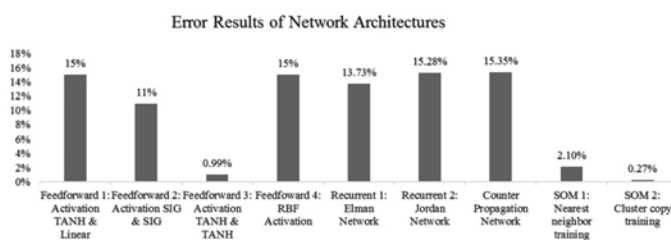


Fig. 8: Average error percentages of four feedforward networks with varying activation functions and two SOM with different training techniques.

Although the TANH feedforward network classified walking speed with an error of less than 1%, the most effective networks for classifying EMG data were SOM neural networks. Both SOM networks had an average error rate of under 2.5%, and the SOM network trained with a cluster copy training algorithm had an error rate of 0.27%. These results give the cluster copy trained SOM in this study an average accuracy percentage of 99.73%, a result which improves upon many machine learning techniques of EMG classification.

TABLE 2
COMPARISON OF CLASSIFICATION RESULTS

Machine Learning Method	Average Accuracy
Adaptive neuro-fuzzy inference system	88.9%
K-NN Clustering	91.25%
Fuzzy K-NN Clustering	92.5%
Discrete Wavelet Transform	96.67%
Dynamic CPN	97.22%
Backpropagation neural net	97.5%
CANFM	98.75
CCT SOM	99.73%

Fig. 9: Comparison of the accuracy percentages obtained from multiple machine learning techniques, including the cluster copy training SOM created in this study.

Table 2 compares the results obtained in this study from the cluster copy training (CCT) SOM with results from different machine learning techniques of EMG classification [6][17][27]. Most of the studies cited in

this table classified gestures or movements, not walking speed. For example, the adaptive neuro-fuzzy inference system (ANFIS), classified input EMG data into one of four different hand movements [6]. Although the classification tasks differ slightly from those in this study, the techniques are similar enough for a comparison to be made between the results.

The results displayed in Table 2 show that the CCT SOM outperforms the other machine learning methods by as much as 10.83%. We can conclude that in creating an algorithm for the classification of walking speed given EMG inputs, a CCT SOM provides the most accurate results, with an equal amount of computational expense as most other machine learning pattern classification algorithms.

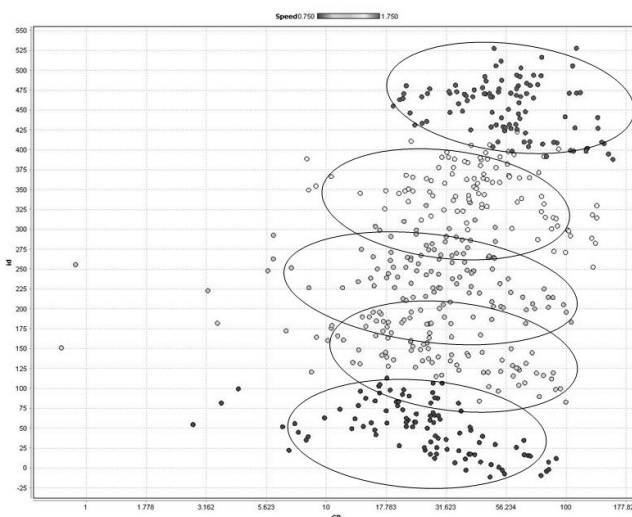


Fig. 10: Visualization of the two-dimensional topology map outputted by the SOM. This visualization does not reflect the exact CCT SOM created in this study, but simply serves for visualization purposes.

6 CONCLUSION

The final goal of this research project is to further research on a more optimal myoelectric prosthetic control scheme that would be able to achieve accuracy in classification tasks while remaining simple enough to be implemented in inexpensive devices. Results from this preliminary study suggest that self organizing maps can be used to accurately classify human movement given EMG input data. Given their promising accuracy in walking speed classification, low computational expense compared to networks that output spatial coordinates, and fast training time, SOMs could potentially form the basis for a more optimal myoelectric control scheme for prosthetics. However, this study utilized a limited data set and did not assess every method of EMG analysis for prosthetics.

Therefore; further work should include a more comprehensive study to cover a wider range of movements with high performance data acquisition and better statistical significance. In addition, the possibility of testing EMG data with different neuroclustering techniques

should be considered, with the ultimate goal of creating a myoelectric control scheme that can successfully integrate machine learning techniques to create a computationally inexpensive and accurate prosthetic device.

REFERENCES

- [1] M. Gandolla, S. Ferrante, D. Baldassini, M. Cotti Cottini, C. Seneci and A. Pedrocchi, *Mediterranean Conference on Medical and Biological Engineering and Computing*, 14th ed. Milano, Italy: NearLab, Department of Electronics, Information, and Bioengineering, 2016, pp. 634-637.
- [2] H. Herr, G. Whiteley and D. Childress, *Cyborg Technology—Biomimetic Orthotic and Prosthetic Technology*. Bellingham, Washington: SPIE Press, 2003, pp. 2-35.
- [3] A. El Gehani, S. Mohamed, H. Busedra and A. Gawedar, "EMG Pattern Recognition System Based on Self-Organizing Map for Myo-Electric Prosthesis", *ICECE*, 2013.
- [4] C. Pattichis, C. Schizas and L. Middleton, "Neural network models in EMG diagnosis", *IEEE Transactions on Biomedical Engineering*, vol. 42, no. 5, pp. 486-496, 1995.
- [5] G. McGimpsey and T. Bradford, *Limb prosthetics services and devices*, 1st ed. Worcester, Massachusetts: Bioengineering Institute Center for Neuroprosthetics, 2008.
- [6] H. Jahani Fariman, S. Ahmad, M. Hamiruce Marhaban, M. Ali Jan Ghasab and P. Chappell, "Simple and Computationally Efficient Movement Classification Approach for EMG-controlled Prosthetic Hand: ANFIS vs. Artificial Neural Network", *Intelligent Automation & Soft Computing*, vol. 21, no. 4, pp. 559-573, 2015.
- [7] A. Balbinot, A. Jnior and G. Favieiro, "Decoding Arm Movements by Myoelectric Signal and Artificial Neural Networks", *ICA*, vol. 04, no. 01, pp. 87-93, 2013.
- [8] D. Farina, Ning Jiang, H. Rehbaum, A. Holobar, B. Graimann, H. Dietl and O. Aszmann, "The Extraction of Neural Information from the Surface EMG for the Control of Upper-Limb Prostheses: Emerging Avenues and Challenges", *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 22, no. 4, pp. 797-809, 2014.
- [9] M. Seyedali, J. Czerniecki, D. Morgenroth and M. Hahn, "Co-contraction patterns of trans-tibial amputee ankle and knee musculature during gait", *Journal of NeuroEngineering and Rehabilitation*, vol. 9, no. 1, p. 29, 2012.
- [10] M. Reaz, M. Hussain and F. Mohd-Yasin, "Techniques of EMG signal analysis: detection, processing, classification and applications", *Biol. Proced. Online*, vol. 8, no. 1, pp. 11-35, 2006.
- [11] A. Au and R. Kirsch, "EMG-based prediction of shoulder and elbow kinematics in able-bodied and spinal cord injured individuals", *IEEE Transactions on Rehabilitation Engineering*, vol. 8, no. 4, pp. 471-480, 2000.
- [12] C. Vaughan, B. Davis and J. O'Connor, *Dynamics of human gait*. Champaign, Ill.: Human Kinetics Publishers, 1992.
- [13] A. Hof, H. Elzinga, W. Grimmius and J. Halbertsma, "Detection of non-standard EMG profiles in walking", *Gait and Posture*, vol. 21, no. 2, pp. 171-177, 2005.
- [14] J. Draye, D. Pavisic, G. Cheron and G. Libert, "Dynamic recurrent neural networks: a dynamical analysis", *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, no. 5, pp. 692-706, 1996.
- [15] G. Cheron, J. Draye, M. Bourgeois and G. Libert, "A dynamic neural network identification of electromyography and arm trajectory relationship during complex movements", *IEEE Transactions on Biomedical Engineering*, vol. 43, no. 5, pp. 552-558, 1996.
- [16] T. Kohonen, "The self-organizing map", *Neurocomputing*, vol. 21, no. 1-3, pp. 1-6, 1998.
- [17] H. Huang, Y. Liu, L. Liu and C. Wong, "EMG Classification for Prehensile Postures Using Cascaded Architecture of Neural Networks with Self-Organizing Maps", 2003 *IEEE International Conference on Robotics & Automation*, 2003.
- [18] C. Christodoulou and C. Pattichis, "Unsupervised pattern recognition for the classification of EMG signals", *IEEE Transactions on Biomedical Engineering*, vol. 46, no. 2, pp. 169-178, 1999.
- [19] Gunther and S. Fritsch, "neuralnet: Training of Neural Networks", *The R journal*, vol. 2, no. 1, pp. 30-38, 2010.
- [20] J. Heaton, *Programming neural networks with Encog3 in Java*.
- [21] L. Fausett, *Fundamentals of neural networks*. Englewood Cliffs, NJ: Prentice-Hall, 1994
- [22] C. Igel and M. Hsken, "Empirical evaluation of the improved Rprop learning algorithms", *Neurocomputing*, vol. 50, pp. 105-123, 2003.
- [23] A. Nicholson, "A Beginner's Guide to Recurrent Networks and LSTMs - Deeplearning4j: Open-source, distributed deep learning for the JVM", *Deeplearning4j.org*, 2016. [Online]. Available: <http://deeplearning4j.org/lstm.html>. [Accessed: 01- Aug- 2016].
- [24] J. Szkola, K. Pancerz and J. Warchol, "Recurrent Neural Networks in Computer-Based Clinical Decision Support for Laryngopathies: An Experimental Study", *Computational Intelligence and Neuroscience*, vol. 2011, pp. 1-8, 2011.
- [25] M. Dweib, "Optimal Architecture Setting of Learning Parameters Based on Counter Back-Propagation Neural Networks", *ARPN Journal of Science and Technology*, vol. 5, no. 12, pp. 616-619, 2015.
- [26] J. Heaton, "Encog Library of Interchangeable Machine Learning Models for Java and C#", *Journal of Machine Learning Research*, vol. 16, pp. 1243-47, 2015.
- [27] E. Gokgoz and A. Subasi, "Comparison of decision tree algorithms for EMG signal classification using DWT", *Biomedical Signal Processing and Control*, vol. 18, pp. 138-144, 2015.

Improving performance of automatic program repair using learned heuristics

Liam Schramm, Jugal Kalita

Abstract—Automatic program repair offers the promise of significant reduction in debugging time. Early generate-and-test systems, such as the genetic programming method GenProg, have had problems creating high-quality patches due to overfitting to the test suite. Recent efforts such as SPR and Prophet demonstrate that including external knowledge about the nature of correct repairs dramatically improves results. SearchRepair demonstrates that including semantic constraints can greatly improve patch quality, but has a high computational cost. Combining Prophet’s learning techniques with SearchRepair’s semantic constraints allows for a method that is both fast and accurate. This paper proposes a modification to SearchRepair that uses a fast classifier to quickly identify good candidate patches. We use a random forest to quickly classify whether two pieces of code are similar, allowing the search to focus only on the best patch candidates. We report 96% accuracy on this classification task, which is enough to greatly improve search speed. We then train another forest on data of whether or not patches were correct, and use this to filter for patches with a high likelihood of success.

Keywords— Automatic program repair, Machine learning, Semantic search.

I. INTRODUCTION

Software debugging is a tedious, expensive, and manual process. The majority of the cost of most software projects is spent on software maintenance, and the majority of the cost of software maintenance is debugging [3]. Automated program repair offers the promise of dramatically lowering or even eliminating these costs. However, the fledgling field still struggles with making tools that are practical enough to reach the level of popular usage. For an automated repair tool to be successful, it must be meet certain criteria of efficiency, accuracy, and generality.

- **Efficiency:** The ability to create and validate a patch for a given bug in a reasonable amount of time. An automated program repair tool must be fast enough that it is cheaper and more convenient to fix the bug automatically than it is to fix it by hand
- **Accuracy:** The probability that the patch proposed for a given bug is correct. Although we would like to be confident enough in the algorithms recommendations that they would not need human oversight, this goal seems far off. At present, a more reasonable goal is that a correct patch is high enough on the list of recommended patches that a human can easily pick it out.
- **Generality:** The range of bugs a repair algorithm is able to address. While tools that address specific types of bugs may be adept in their particular field, there are simply too many different types of bugs for addressing

them with individual programs to be practical. Methods that use certain preset types of modifications report that no one modification ever accounts for a large fraction of total repairs [10], [12], [4]. For automated program repair to become highly useful in a real-world environment, it must be general enough to fix a wide range of bugs

Several methods exist for automatic program repair, but all of them encounter problems with at least one of the above criteria. Semantic algorithms like DirectFix or Angelix are limited in the bugs they are able to address and sometimes scale poorly [1]. Search-based algorithms such as GenProg and SPR offer the possibility of a wider range of patches, but also are very likely to create patches which pass the test suite, but are ultimately incorrect [1], [4] [3]. One reason that search-based methods have this problem is that the search space is effectively infinite, and within this space, there are many more patches that pass all tests (plausible patches) and are incorrect than there are correct patches [5]. One solution to this problem is to use a fast, learned heuristic to focus the search on patches that look like human-written patches, which are more likely to be correct [2]. Another approach is to introduce semantic constraints and use theorem proving to ensure that undertested functionality is not deleted [8], [13].

In this paper, we combine this fast heuristic search with more rigorous theorem-proving techniques to create a method that is both fast and reliable.

II. BACKGROUND

Automated program repair is the process of patching bugs in a program given a test suite for that program. Currently, there are two common approaches to this problem: generate-and-validate, and semantic analysis. Generate-and-validate solutions create a large number of candidate patches, and then evaluate them one-at-a-time until an optimal candidate has been chosen. This candidate is then tested against the test suite, and if it passes, is accepted. If it is not accepted, a new patch is chosen, and the process is repeated until an acceptable patch is chosen, the search space is exhausted, or the time limit is reached. Semantic analysis works by extracting a repair constraint through symbolic execution. These constraints are then fed to theorem provers that infer a correct patch, which is then inserted.

In 2009, GenProg became the first algorithm to successfully perform automated program repair on large-scale real-world problems. It uses genetic programming to encode possible patches as variations to the source code. Its algorithm works approximately as follows:

- 1) Randomly generate a population of potential patches.
- 2) Run a subset of the test suite on each member of the population. The number of tests each member passes serves as the fitness function.
- 3) Breed the members of the population with the highest fitness.

In the months after GenProg’s publishing, however, it became apparent that the method was not as effective as it had first seemed. Several studies found that GenProg frequently “fixed” bugs by deleting functionality. This is because its fitness function is defined solely by the test suite. Any functionality not covered by the test suite is likely to be deleted if any part of it contributes to buggy behavior [7]. Since the GenProg stops searching once it has found a patch that passes all the tests in the test suite, this pattern of functionality-deletion often contributes to GenProg failing to find patches that are within its search space [5].

This problem can also be viewed as an issue of overfitting. In a sense, GenProg overfit to the test suite by using it as the sole determinant of whether a patch was correct. Thus, it created patches which performed well on the test suite but poorly fit the actual function of the code. Most of the work done in the field since this discovery can be seen as an attempt to avoid overfitting.

Within the generate-and-validate framework, two methods have emerged for improving the quality of patches. The first approach was taken by Prophet and SPR. Although both still create incorrect patches about half the time, they reduce the likelihood of this happening by trying to produce a small number of patches that obey a collection of hand-coded rules [4] [2]. Each hand-coded rule, or schema, covers a specific type of repair. SPR and Prophet first use a target value search to check whether each schema is capable of generating a patch for the given bug. If it is not, the schema is discarded. They then generate a search space of patches from the remaining schemas, and test each of those

Since correct patches are sparse in the space of plausible patches, randomly generating solutions is more likely to produce plausible, incorrect patches than correct patches [5]. The authors believe that their careful construction of patches is less likely to produce incorrect patches because it utilizes more information about the program. Prophet also uses machine learning to select for patches with similar qualities to human patches [2]. This has several advantages. Firstly, it allows Prophet to rank patches without using the test suite, reducing the threat of overfitting. Since applying the learner is also much faster than running the test suite, it also serves as a useful search heuristic. This allows Prophet to focus its search significantly by looking for patches that share features with the correct patches.

The other approach, which was taken by SearchRepair, uses an algorithm that combines the search and the semantic analysis paradigms [8].

- 1) Encode a database of human-written code fragments as satisfiability modulo theories (SMT)
- 2) Locate the bug

- 3) Build a functional description of each fragment that describes its input-output profile
- 4) Search the database for a code fragment that matches the desired input-output profile
- 5) Insert the code fragment and run the test suite

While still being somewhat similar to its ancestor GenProg, SearchRepair has a few changes that help it combat overfitting. Firstly, it copies and pastes larger sections of code. The intuition is that while a single line may behave very differently out of context, a multiline block of code is less likely to do so. It would replace the entire buggy section with a correctly implemented version written by another human developer. Since this replacement block was not overfit to the test suite when it was written, there is less risk of it being overfit here. In addition, SearchRepair uses theorem proving to speed up this search. Although theorem proving is very slow, it is still much faster than running the test suite for every patch.

SearchRepair generated patches with much higher quality than its purely random counterparts, passing 97.3% of tests in an independent test suite as opposed to GenProg’s 68.7%. Although SearchRepair did not fix as many bugs as GenProg, RSRepair, or AE, this may be because these other methods might have fixed a large number of bugs simply by deleting functionality [8], [7].

SearchRepair has major issues with scalability [13]. It was almost two orders of magnitude slower than the second slowest method run on the Introclass benchmark, and three orders of magnitude slower than all other methods. One of the main reasons for this is the way its theorem prover works. Because SearchRepair uses pieces of code from other applications as its patches, it must rename the variables to insert the code. However, it has no a priori way of knowing what the correct variable mapping is, so it must run theorem proving on each possible mapping [13]. This means that the theorem prover must run $n!$ times on each patch, where n is the number of variables. Since SMT satisfiability is already an NP-Hard problem, SearchRepair effectively tries to brute-force an NP-Hard problem (variable mapping) that involves solving another NP-Hard problem (SMT satisfiability) on every iteration. This is, needless to say, very inefficient.

Amazingly, while this process is prohibitively expensive for large numbers of patches, the time taken to test a single patch is not terribly high. Since most code fragments used by SearchRepair have no more than 4 or 5 variables, a single patch can often be tested in less than a second. It is only when large databases of code fragments must be searched that problems arise.

III. METHODS

As stated earlier, both SearchRepair and SPR/Prophet have drawbacks, either in their scalability or in their accuracy. For automatic program repair to become practically viable, a system must both create high-quality patches and be able to run in a reasonable amount of time.

One way to avoid the large computational cost associated with SearchRepair’s semantic search while maintaining high

accuracy is to use a fast heuristic to eliminate poor candidates. Using a heuristic to iterate over all the possible patches in a database lets us pick out those with a high likelihood of success, avoiding the majority of the computation previously required. This heuristic must be independent of variable mappings in order keep its cost from becoming prohibitively expensive. The theorem prover would then run on these selected candidates, ensuring that each proposed patch is indeed of high quality.

Since the cost of using theorem proving is much higher than the cost of using the learner, a high false negative rate is much more acceptable than a high false positive rate. If, for instance, there was a false negative of 80%, we could simply try five times as many patches at fairly low cost. However, if we evaluate 1000 patches and have even a 10% false positive rate, this means we must evaluate 100 patches with theorem proving. Given SearchRepair's current speed, this would take about a minute. However, since the learner has been able to evaluate around 50,000 patches per second, it would be more efficient to take a much larger database of patches and use a learner with a very low false positive rate. Even if the learner has a false negative rate of over 50 percent, we can easily replace these these cases by just searching more patches.

To create such a heuristic, we begin with the feature extractor proposed by Long and Rinard in their Prophet paper [2]. This method creates a list of atoms (variables or constants) and what operations the program uses on them. For instance, if a program checks `if(a > c && a < b)`, then assigns `median = a`, the feature extractor would record `a: lessthan, greaterthan, < assign, R>`. The fact the `a` was on the right side of the assignment is recorded because assigning and being assigned to are very different operations.

The complete list of features is enumerated below.

- 1) `var`: True if the atom is a variable
- 2) `const0`: True if the atom is a constant equal to 0
- 3) `constn0`: True if the atom is a constant not equal to 0
- 4) `cond`: True if the atom is in a conditional
- 5) `iff`: True if the atom is in the body or the conditional of an if statement
- 6) `prt`: True if the atom is printed
- 7) `loop`: True if the atom is in the body or the conditional of a loop
- 8) `EQ`: True if the atom is in a statement with a `==`
- 9) `NEQ`: True if the atom is in a statement with a `!=`
- 10) `ret`: True if the atom is returned
- 11) `plus`: True if the atom is in an addition statement
- 12) `times`: True if the atom is in a multiplication statement
- 13) `minus_l`: True if the atom is on the left side of a subtraction operator
- 14) `divided_l`: True if the atom is on the left side of a division operator
- 15) `minus_r`: True if the atom is on the right side of a subtraction operator
- 16) `divided_r`: True if the atom is on the right side of a division operator
- 17) `increment`: True if the atom is incremented

- 18) `decrement`: True if the atom is decremented
- 19) `ASSIGN_L`: True if a value is assigned to the atom
- 20) `ASSIGN_R`: True if the atom is assigned to another variable
- 21) `LESS_THAN`: True if the atom is in a statement with a `<`
- 22) `GREATER_THAN`: True if the atom is in a statement with a `>`
- 23) `LESS_EQ`: True if the atom is in a statement with a `<=`
- 24) `GREATER_EQ`: True if the atom is in a statement with a `>=`

These features are calculated for a given line(C), the three lines above it(P), and the three lines below it(N). The complete feature vector for a line in a program includes the set of features for C, P, and N for each atom in C.

Since the goal is to evaluate whether a given patch will work for given buggy section, we append the feature vectors of the buggy code and the proposed patch to create a new vector, and feed this vector to the learner. However, this gives rise again to the issue of variable mappings. In the Prophet feature extractor, the features of an atom in the buggy program are explicitly encoded alongside the features of the corresponding variable in the patched program. Since encoding this mapping, even implicitly, would defeat the point of a mapping-free heuristic, we randomize the order in which atoms are encoded for every set of features, preventing the learner from finding mapping-dependent patterns.

We train a random forest to classify patches as either correct or incorrect. Then, we use this classifier as a filter for what patches should be evaluated by the theorem prover. The modified algorithm is as follows.

- 1) Encode a database of human-written code fragments as satisfiability modulo theories (SMT)
- 2) Locate the bug
- 3) Build a functional description of each fragment that describes its input-output profile
- 4) Extract features of the buggy code
- 5) Extract features of each fragment in the database
- 6) Combine the buggy features and the fragment's features into a single patch feature vector
- 7) Apply the learner to each patch feature vector in the database. If the patch feature vector is classified as correct, add the corresponding code fragment to a second database
- 8) Search the new, filtered database for a code fragment that matches the desired input-output profile
- 9) Insert the code fragment and run the test suite

The following example serves to demonstrate the new algorithm. Suppose we have the following bug from the IntroClass benchmark.

```

/**/
#include <stdio.h>
#include <math.h>
int main(void)
{
    int int1, int2, int3, med;

```

```

printf("Please enter 3 numbers separated
by spaces > ");
scanf("%d %d %d", &int1, &int2, &int3);

if (((int1 < int2) && (int1 > int3)) ||
    ((int1 < int2) && (int1 > int3)))
    med = int1;
else if (((int2 < int1) && (int2 >
int3)) || ((int2 < int3) && (int2 >
int1)))
    med = int2;
else if (((int3 < int1) && (int3 > int2))
    || ((int3 < int2) && (int3 > int1)))
    med = int3;

printf("%d is the median\n", med);
return 0;
}

```

Since this program uses `<` and `>` operators instead of `<=` and `>=` operators, it will fail in the case that two or more of the numbers are equal.

Suppose then, that the modified SearchRepair's search space includes the following three patches

Patch 1

```

while ((status = scanf("%c", &it)) != EOF &&
it != '\n')
sum = (sum + (long) it) % 64;
sum = sum + (long) ' ';
printf("Check sum is %c\n", (char) sum);

```

Patch 2

```

for(i=flag1-1; i>=flag2; i--)
{
if(flag2==1 && i==1)
printf("-");
printf("%c\n", digit[i]);
}

```

Patch 3

```

if((a >= b && a <= c) || (a >= c && a <= b))
median = a;
if((b >= a && b <= c) || (b >= c && b <= a))
median = b;
else
median = c;
printf("%d is the median\n", median);

```

Once the database has been constructed, the bug located, and the functional descriptions constructed, the algorithm begins the machine learning section. First, it constructs the feature vector for buggy section of code and for each of the patches. Then, it concatenates the two vectors into a single vector which is fed to the learner. In this case, let's presume that the learner classifies patch 2 and patch 3 as viable, but patch 1 as not viable. Patches 2 and 3 are added to the temporary database. SearchRepair then fetches patch 2 from the database and runs the theorem prover on it. Since it doesn't match the functional specifications given by the test suite, it is discarded. SearchRepair fetches patch 3 from

the database and applies the theorem prover to it as well. Since patch 3 fits the semantic constraints, it is accepted, and its variables are replaced with the variables from the buggy program. Finally it is tested against the test suite. It passes, and the final patch is returned.

```

/**/
#include <stdio.h>
#include <math.h>
int main(void)
{
int int1, int2, int3;
printf("Please enter 3 numbers separated
by spaces > ");
scanf("%d %d %d", &int1, &int2, &int3);

if((int1 >= int2 && int1 <= int3) || (int1
>= int3 && int1 <= int2))
med = int1;
if((int2 >= int1 && int2 <= int3) ||
(int2 >= int3 && int2 <= int1))
med = int2;
else
med = int3;
printf("%d is the median\n", med);
return 0;
}

```

IV. EVALUATION

Using these feature vectors, we trained a random forest to classify pairs code snippets based on whether they were from programs that were trying to do the same thing. The intuition is that a program with similar functionality is far more likely to provide a correct patch than a program with very different functionality. These code snippets were sampled from the IntroClass benchmark. A pair was counted as a positive case if the two snippets were from programs submitted for the same assignment. Otherwise, they were counted as a negative case. The positive cases were sampled at a higher rate to make up for the asymmetry in the size of the classes.

To evaluate how effective different machine learning methods were for this set of features, we took a subset of the data and tried an array of machine learning techniques to see which produced the best results

Algorithm	Training Time	Accuracy
SGD	13.96 s	51.297%
Naive Bayes	.21 s	55.951%
Bayes Net	.76 s	56.002%
Random Forest	14.07 s	86.165%
J48	27.22 s	75.534%
AdaBoost	6.58 s	56.738%
K-nearest neighbors	n/a	81.587%
SMO	540.51	54.9593

TABLE I: Learner results

Based on these results, we concluded that a Random Forest would be the most effective learner for this application. For this reason, the rest of our experiments are conducted using a random forest.

Using the full dataset, we train a random forest with 100 trees, run for 100 iterations, and test it using 10-fold cross-validation. The forest was correctly able to classify 96.17% of the test cases. The false positive rate (classified as being from the same assignment, but really from different assignments) was 5.4%. The false negative rate (classified as being from different assignments, but really from the same assignment) was 2.4%. For this application, this is a very satisfactory result. Since the vast majority of the patches are from programs with a different purpose than the buggy program, being able to rule out almost 95% of these obviously bad candidates will significantly increase the search speed.

Once it was demonstrated that this method was effective at the code similarity task, it was applied to the patch recognition task as well. For this, we used a random forest and the same training data as Prophet [2]. We classify a patch as correct if and only if it is the developer-written patch. Although this is not necessary correct, very few copy-pasted code segments will produce correct results, whereas copying and pasting the developer patch into the buggy section always will. We did not use actual SearchRepair patches because it would be computationally infeasible to create a large database of patch data using such a method. Thus, pairs of bugs and their respective developer-written patches are considered positive cases, and other pairs are considered negative cases.

We found that the learner had a 66% false negative rate and a 0.11% false positive rate. It was able to evaluate a set of 118,000 proposed patches in four seconds (Not including time to write the file). Of the patches classified as correct by the learner, 20 out of 139, or about one in seven were in fact the developer patches. This is a huge increase in the density of correct patches. The space of potential patches generated for the learner contained about 1.6 correct patches per thousand. Filtering to one in seven is a dramatic improvement. Although most of the correct patches were discarded, the fact that the learner was able to search the space so quickly allows us to increase the total number of correct patches simply by expanding the state space.

V. FUTURE WORK

Now that the learner has been shown to be effective, the next step would be to make the database adaptive as well. In the current implementation of SearchRepair, the patch database is scavenged more or less at random from a parent application or applications. Some of these patches are likely to be much more successful. For instance, a code snippet with an if statement checking whether or not an integer is equal to zero will likely be a far more useful patch than a section of code implementing a complex mathematical formula. If the formula code is used infrequently enough, then the cost of checking its relevance for every bug will outweigh the benefits of keeping it stored. If such a snippet is repeatedly not used, then it should be deleted from the database and replaced with another snippet with a better likelihood of repairing patches. Similarly, highly redundant snippets are

possible in the current implementation. By keeping track of both how often a given snippet fails and how much its fixes overlap with those of other snippets, the database itself can be made adaptive. This would allow the possibility of establishing a general set of patches that cover a wide range of bugs. Other authors have tried similar techniques with good results. For instance, in *History Driven Program Repair*, the authors use a graph-based learner to learn general patch types from developer patches of real programs [12]. This learner is used to guide the creation of future patches, just like here the success of code snippets would guide their future use.

VI. CONCLUSIONS

Automated program repair holds a great deal of promise, but it is rapidly becoming apparent that purely random methods are not sufficient for high-quality patches. Both patch synthesis methods like SPR and Prophet, and semantic constraint search-based methods such as SearchRepair have proven effective, but each has drawbacks as well. Combining Prophet's fast feature-based search with SearchRepair's slower, more reliable constraint-solving allows us to produce a method that is both fast and accurate.

ACKNOWLEDGEMENT

The authors would like to acknowledge the help of Claire Le Goues and Afsoon Afzal for their implementation of SearchRepair, as well as Kristen Justice for her guidance in this project.

REFERENCES

- [1] S. Mechtaev, Y. Jooyong, and R. Abhik R., Angelix: Scalable multiline program patch synthesis via symbolic analysis, in *Proceedings of the 38th International Conference on Software Engineering.*, New York, NY, 2016, pp. 691-701.
- [2] F. Long and M. Rinard, Automatic patch generation by learning correct code, in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 2016, pp. 298-312.
- [3] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, A Systematic Study of Automated Program Repair: Fixing 55 out of 105 bugs for \$8 Each. *International Conference on Software Engineering (ICSE)*. 2012, pp. 3-13.
- [4] F. Long and M. Rinard, Staged program repair with condition synthesis, in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* 2015, pp. 166-178.
- [5] F. Long and M. Rinard, An analysis of the search spaces for generate and validate patch generation systems. *Proceedings of the 38th International Conference on Software Engineering.* 2016, pp.702-713.
- [6] C. Le Goues, S. Forrest, and W. Weimer, Current challenges in automatic software repair. *Software Quality Journal* vo. 21, no. 3, pp. 421-443, 2013.
- [7] Z. Qi, F. Long, S. Achour, and M. Rinard, An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. in *Proceedings of the 2015 International Symposium on Software Testing and Analysis.* 2015, pp. 24-36.
- [8] Y. Ke, K. Stolee, C. Le Goues, and Y. Brun, Repairing Programs with Semantic Code Search. In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference.* 2015, pp. 295-306.
- [9] M. Richardson and P. Domingos, Markov logic networks. *Machine learning.* 2006. vo. 62, no.1-2, pp.107-136.
- [10] D. Kim, J. Nam, J. Song, and S. Kim, Automatic patch generation learned from human-written patches. In *Proceedings of the 2013 International Conference on Software Engineering.* 2013, pp. 802-811.

- [11] C. Le Goues, N. Holtshulte, E.K. Smith, Y. Brun, P. Devanbu, S. Forrest and W. Weimer, The ManyBugs and IntroClass benchmarks for automated repair of C programs. *Software Engineering, IEEE Transactions*, 2015 vo. 41, no. 12, pp.1236-1256.
- [12] X. Le , D. Lo, and C. Le Goues, History Driven Program Repair. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, vo. 1, pp. 213-224.
- [13] Y. Ke. "An automated approach to program repair with semantic code search." M.S. thesis, CS Dept., ISU, Ames, IA, 2015.

An Open Set Protocol for Improving Malware Classification in Intrusion Detection

Cora A. Coleman*, Ethan M. Rudd†, and Terrance E. Boul†‡

*New College of Florida, Sarasota, Florida

†‡University of Colorado, Colorado Springs, Colorado

Vision and Security Technology (VAST) Lab

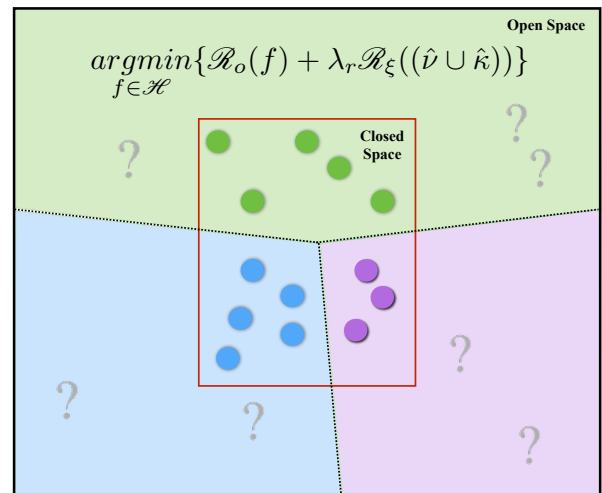
Email: *cora.coleman14@ncf.edu, †erudd@vast.uccs.edu, ‡tboul†@vast.uccs.edu

Abstract—Confidently distinguishing a malicious intrusion over a network is an important challenge. Most intrusion detection system evaluations have been performed in a closed set protocol in which only classes seen during training are considered during classification. Thus far, there has been no realistic application in which novel types of behaviors unseen at training – unknown classes as it were – must be recognized for manual categorization. This paper comparatively evaluates malware classification using both closed set and open set protocols for intrusion recognition on the KDD’99 dataset. In contrast to much of the previous work, we employ a fine-grained recognition protocol, in which the dataset is loosely open set – i.e., recognizing individual intrusion types – e.g., “sendmail”, “snmp_guess”, ..., etc., rather than more general attack categories (e.g., “DoS”, “Probe”, “R2L”, “U2R”, “Normal”). We also employ two different classifier types – Gaussian RBF kernel SVMs, which are not theoretically guaranteed to bound open space risk, and W-SVMs, which are theoretically guaranteed to bound open space risk. We find that the W-SVM offers superior performance under the open set regime, particularly as the cost of misclassifying unknown classes at test (i.e., classes not present in the training set) increases. Results of performance tradeoff with respect to cost of unknown as well as discussion of the ramifications of these findings in an operational setting are presented.

Index Terms—Intrusion Detection, Open Set, Malware, Recognition, Machine Learning, Support Vector Machines, Intrusion Detection, Open Set, Malware, Recognition, Machine Learning, Support Vector Machines

1. Introduction

Intrusion detection systems (IDS) seek to recognize anomalies and attacks in networks. Thus far, the problem of distinguishing between normal and malicious activities has approached classification with a closed world assumption. A closed world can consider at classification time only instances from classes that were available in the training set [1]. Applications to a real world environment where both pre-seen known classes must be recognized and novel classes must be labeled as “unknown”, have heretofore not been taken into consideration. Although some authors



Where $\mathcal{R}_o(f)$ is the open space risk, λ_r is the regularization constant, and $\mathcal{R}_\xi((\hat{v} \cup \hat{k}))$ is the empirical risk function.

Figure 1. In contrast to unrealistic closed-set benchmark settings, this paper tackles the open set intrusion recognition problem. Consider three types of intrusive behavior shown as green, blue, and purple classes, with decision boundaries shown as linear classifiers. These classifiers partition the hypothesis space such that each class of data has unbounded support; thus a sample from a novel unknown class – i.e., far from known data (shown as a question mark) is classified as belonging to one of the three classes under this regime. Instead it should be brought to the attention of the system operator by marking it with an “unknown” label. Open set recognition seeks to bound the classification decisions by the support of known data (shown by the red square). In this closed space, the classification decision is reasonable, and the decision made by the classifier is used as the classification decision. Beyond this bound (e.g., in open space), an effective open set classifier recognizes that samples are not supported by known training data and the classification is marked as “unknown”.

defend a testing methodology in real environments, most advocate an evaluation procedure in experimental benchmark settings [2]. Both techniques have benefits and con-

sequences. One advantage of evaluating real environments is that the traffic is amply realistic and unknown attacks are present, but the effectiveness of the system can only be evaluated in a post-mortem sense, due to the risk of potential attacks: some form of benchmarking is required, but evaluation protocols to date are highly artificial, often with unrealistic distributions of attack and normal data. Moreover, benchmark evaluations are almost always constructed under the premise that all classes of attack and normal behavior present in test will have been seen in train.

This benchmark assumption is unrealistic. Often, novel classes of attacks will emerge precisely to defeat the intrusion detection system. The best recourse that the system can have is to mark these attacks as unseen and flag them for human inspection and training of a newer version of the classifier (or performing dynamic updates in the incremental context). In this paper, we provide the first open set intrusion detection analysis by performing *fine-grained* recognition of attack types within the KDD'99 dataset, wherein not all classes in the test set are seen at training.

This dataset is mostly closed-set, where all instances are classified as either normal or malicious.

Research in intrusion detection has been mostly focused on anomaly-based and misuse-based detection techniques [3]. Misuse-detection is generally favored in commercial products due to its high accuracy. On the other hand, anomaly-detection is conceived as the more powerful method in academic research, due to its theoretical potential for detecting novel attack types. Developing a robust intrusion detection system follows performing either form of misuse or anomaly detection, or a combination of them on network traffic data, which can involve employing a customized machine learning algorithm – the algorithm's goal being to learn the general behavior of the data set so as to be able to distinguish between normal and malicious activities. The novel open-set approach to intrusion recognition that we present in this paper combines both anomaly detection and discriminative misuse-detection via one unified open set classifier (the W-SVM).

Malware classification can illuminate how malicious software attacks devices, the level of threat it poses to those devices, and how to defend against it. Most intrusion recognition techniques, as surveyed in [2], [4], and [5], assume that all classes seen at classification time are also present in the training set and yield recognition accuracy only for a determined closed set of classes. Realistically, recognition has to consider three basic categories of classes: known classes, known unknown classes, and unknown unknown classes [6]. Known classes are those with distinctly labeled positive training examples, which also serve as negative examples for other known classes. Known unknown classes are those with labeled negative examples. Known unknowns are not necessarily grouped into meaningful categories. The inclusion of known unknown classes results in models generated with an explicit "other class," or a detector that is trained with unclassified negatives. Finally, unknown unknown classes are classes unseen in training.

We seek to show how multi-class recognition of un-

known data points can be extended to malware classification through an evaluation comparison of support vector machines (SVM). We chose the KDD'99 dataset as our benchmark to perform malware classification using a Radial Basis Function (RBF) kernel SVM and the Weibull-calibrated SVM (W-SVM) algorithm. By classifying on fine-grained attack types within the KDD'99 dataset, we implicitly transform the problem to an open set one, which we evaluate under two protocols: a closed set evaluation protocol and an open set evaluation protocol. While the RBF SVM is not mathematically guaranteed to bound open-space risk (amount of unsupported unknown hypothesis space labeled as a class in training), we can attempt to do so by performing thresholding on Platt-calibrated probability estimates; the W-SVM by contrast is mathematically guaranteed to bound open space risk, but the tightness of the bound can vary. In analogous contexts in object recognition and OCR, Scheirer et al. found that the W-SVM performs significantly better in an open set regime than other state-of-the-art for the same tasks [6].

Our contributions are as follows:

- We evaluate the KDD'99 dataset under an open set framework and demonstrate how open set classifiers can yield gains in accuracy.
- We introduce a novel analysis on the cost of unknown misclassifications that illustrates how a choice for an open set or closed set classifier when evaluating the data can impact the cost.
- We perform fine-grained classification on the KDD'99 dataset as opposed to using the four rough categories most other evaluations have followed.
- We analyze dataset balance and evaluate its effects on classification.

2. Related Work

Although the KDD'99 dataset is not the best dataset for intrusion detection evaluation, the ubiquitous nature of the KDD'99 dataset in intrusion detection research is our motivation for selecting it. A year-wise distribution of datasets used for intrusion detection experiments can be found in [4]. Among them, and ranked as the most widely used, is the KDD'99 dataset. This demonstrates how widely-used the KDD'99 dataset is for intrusion detection, despite its limitations. KDD'99 was constructed based on the data captured in the DARPA'98 IDS evaluation program. The dataset is composed of 4 gigabytes of compressed raw (binary) tcpdump data collected from 7 weeks of simulated attacks on network traffic. The training dataset contains about 4,900,000 single connection vectors, each of which contains 41 features and is labeled either as "normal" or an "attack." The attacks simulated on the network can be categorized into the following: Denial of Service Attack (DoS), User to Root Attack (U2R), Remote to Local Attack (R2L), and Probing Attack.

Important deficiencies in the KDD'99 data set include a huge number of redundant records. Tavallae et al. analyzed

the KDD'99 dataset and discovered that 78% and 75% of the records are duplicated in the train and test set, respectively [3]. Within the train set, these redundancies will cause learning algorithms to be biased towards more frequent records, which in turn hinder it from learning infrequent records. Infrequent records are typically more harmful to networks, for example User to Root Attacks (U2R). Within the test set, redundancies will cause the evaluation results to be biased towards methods that have better detection rates on the more frequent records. In the Approach section of this paper we discuss the procedure we took to normalize the KDD'99 dataset before conducting our experiments.

2.1. Open Set Recognition

Differentiating *intrusion detection* and *intrusion recognition* is important for understanding the applications of this paper. Detection generally demands the identification of anomalous behavior, whereas for recognition we assume there are some classes we can recognize in a much larger space of things we do not recognize. Open set recognition can be defined as a real world problem for which unknown inputs and incomplete knowledge are present in multi-class recognition. When considering open set recognition, many of the assumptions behind traditional statistical learning, Bayesian models, and generative and discriminative models oftentimes do not hold. Despite this, open set recognition can be adapted to provide probabilities for thresholding decisions where those decisions depend on the validity and shape of those probabilities [6].

In the scope of computer vision, [7] formalized the open set recognition problem as a risk-minimizing constrained functional optimization problem. They introduced a novel "1-vs-Set Machine" that defines a decision space from the marginal distances of a 1-class or binary SVM using a linear kernel in order to support better generalization and specialization in a manner that remains consistent with the open set problem definition. The experiments performed for object recognition and face verification reveal that the 1-vs-Set Machine is highly effective at improving accuracy when compared to 1-class and multi-class SVMs under the same test regime.

In 2014, Scheirer et al. addressed the general idea of open space risk limiting classification in order to accommodate non-linear classifiers in a multi-class setting [6]. In examining the problem of open set recognition, they proposed a model incorporating an open space risk term that could account for the space beyond the reasonable support of known classes. Statistical extreme value theory (EVT) was used to develop a novel approach to probability estimation for SVMs by observing that the distributions of score tails near the decision boundary follow EVT distributions. This novel approach is the Weibull-calibrated SVM (W-SVM), which combines Compact Abating Probability (CAP) models in [6] with EVT for improved multi-class open set recognition. Within a CAP model, probability of class membership wanes as points move from known data to open space, which accounts for the unknown unknowns

without the need to explicitly model them. The experimental results show the strong impact of openness on SVMs, where applying open set recognition requires thresholding on estimates that are robust to unknown classes and decay away from training data. Nevertheless, with very limited sampling in training for a class with large variation in its feature space, it may not always be possible to fit a good Weibull model to the data.

More recently, in 2016 Rudd et al. surveyed malicious stealth technologies and existing autonomous countermeasures [1]. Their findings suggest that while machine learning has potential for generic and autonomous solutions, several flawed assumptions fundamental to most recognition algorithms inhibit a direct mapping between the stealth malware recognition problem and a machine learning solution. The closed world assumption was the most notable of these flawed assumptions. Unseen classes at classification time exist for truthful intrusion recognition tasks, and neither all variations of malicious code nor all variations of harmless behaviors can be known apriori. Finally, Rudd et al. introduce an open set recognition framework to be incorporated into existing intrusion recognition algorithms. The central point being that there is no need to discard closed set algorithms in order to manage open space risk, given that they are combined with open set recognition algorithms. Closed set techniques are regarded as sufficient solutions when they are well supported with training data, however, open set algorithms are necessary to draw meaning from closed set decisions. Thus, the open set problem can be approached by using an algorithm that is inherently open set for novelty detection and that rejects any closed set decision as unknown if its support is below the open set threshold [1].

3. Approach

Support vector machines (SVMs) were formally proposed in 1992 with a publication by Boser, Guyon, and Vapnik [8]. SVMs first map the input vector into a feature space to then obtain the optimal separating hyper-plane in the feature space. In addition, the decision boundary, i.e., the separating hyper-plane, is determined by the support vectors rather than the entirety of training samples and as a result is extremely resilient to outliers.

SVMs were first designed to perform linear classification, fit to a *training set* of binary labeled data, but they have been extended to both multiclass problems via one-vs-one and one-vs-rest formulations, nonlinear problems via kernelization, and one-class formulations [9].

For a kernelized SVM, the classification decision for sample x' is given by

$$h(x') = \sum_{i=1}^M \alpha_i y_i K(x_i, x') + b, \quad (1)$$

where M is the number of support vectors, α_i is the Lagrange multiplier and label corresponding to the i th support vector, x_i is the i th support vector in the input space, K is the kernel function, and b is the bias [9].

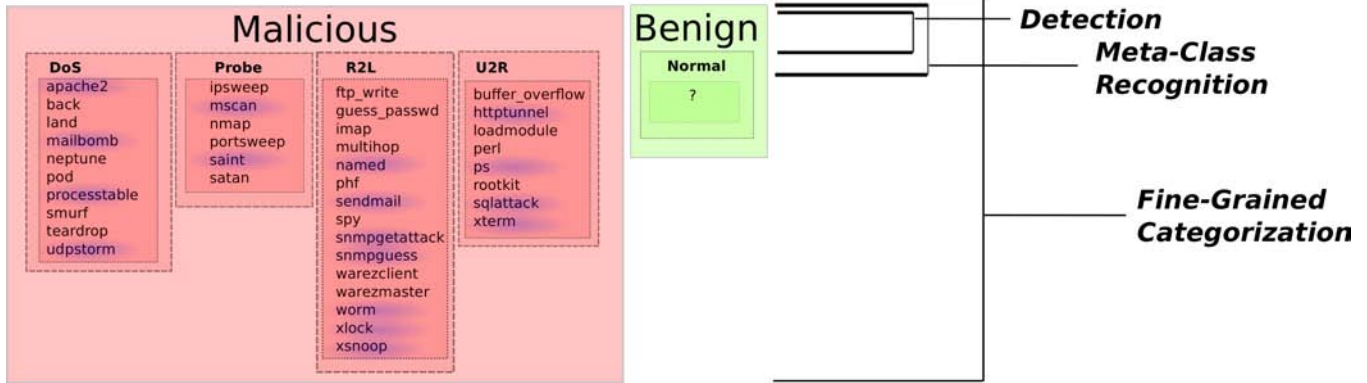


Figure 2. When evaluating the KDD'99 dataset, train and test datasets can be considered at different granularities. Most authors evaluate either the intrusion detection setting – i.e., binary classification of samples as *malicious* or *benign*. While some authors classify different samples by metatype (DoS,Probe,R2L,U2R,Normal), which is a closed-set protocol, we perform *fine-grained* evaluation of the KDD'99 dataset, in which novel types of attacks are present in the test set. Attack classes that are present in the test set that are not present in the train set for the original KDD'99 dataset are shown in blue (this is prior to changes from pre-processing discussed in Sec. 4.1).

Standard SVMs do not provide a calibrated posterior probability as the classification output, which limits the post-processing of the evaluation. Platt empirically found that training an SVM then performing a maximum likelihood fit of a sigmoid on distances of training samples from an SVM's decision boundary, yields a good discriminative estimate of probability of inclusion with respect to a class of interest [10]. The normalization is parameterized as

$$\sigma(x; c, t) = \frac{1}{1 + \exp(-c(x - t))}, \quad (2)$$

where c , the *temperature* of the sigmoid and t , the translation parameter, are learnt by Maximum Likelihood Estimation (MLE). However, Platt calibration this does not bound open space risk in the binary setting, due to the infinite extent of a sigmoid. When averaged over multiple classes in a 1-vs-1 multiclass regime, Platt calibration *can* bound open space risk, but there are still no guarantees. Even with the assumption that all classes are mutually exclusive, the unknown unknown classes prohibit the use of the law of total probability that underlies Bayes' theorem [6]. Open set recognition cannot only use the maximum a posteriori probability (MAP) estimate over the known classes as the best solution because MAP estimation requires the full posterior distribution, and a consideration of all classes. The relevance of only the known classes is insufficient.

Scheirer et al. note that, while it is well known that one-class models are typically less effective than binary machines, the decision score of a binary SVM is not a canonical sum [6]. However, the decision score can still be useful because improved probabilities will generally result in tighter bounds about the class of interest. After collecting all of the positive coefficients into one sum, and all of the negatives into a second sum, and splitting the bias between them, one can view the SVM as applying a decision rule on whichever is more similar. Effectively, this technique combines both positive and negative evidence. By only working

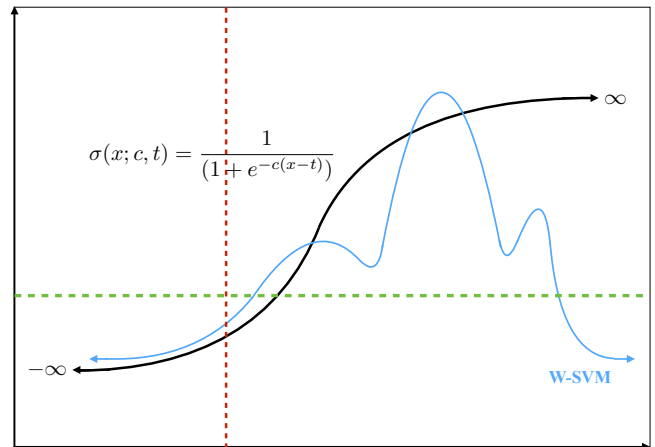


Figure 3. Let red correspond to the decision boundary. Platt calibration (sigmoidal) with respect to distance from the decision boundary of an RBF SVM does not bound open space risk because when thresholded (green), the calibration still labels unbounded space with approximately 100% probability. Probabilities of inclusion returned from a Weibull-calibrated W-SVM (e.g., blue) abate to zero with distance from the actual data, and thus provably bound open space risk.

with the positive or the negative data, one can obtain a model for nicely bounded results from a binary SVM that can be used in addition to the one-class probabilities. This model is the Weibull-calibrated SVM (W-SVM). To bound open space risk, the W-SVM uses the decision output from an EVT-calibrated one-class density estimator RBF SVM. To bound open space risk, there is a constraint that α_i be non-negative $\forall i$; thus not all formulations of one-class SVMs will do because this constraint is vital to the *compact abating* property in Theorem 2 of [6].

For the purposes our experiments, we use a 1-vs-Rest implementation for the W-SVM from [6], and a 1-vs-1 default implementation from Scikit-learn for the conventional

RBF SVM. A Gaussian RBF kernel was selected for all of the SVMs. Note that Scikit-learn default implementation uses an averaging of calibrated Platt probabilities, which make many class pairs (1-vs-1) more resilient to an open set protocol.

<i>Training</i>		<i>Testing</i>	
Class	Instances	Class	Instances
1	53	1	1302
2	2	3	17
4	8	4	3
6	1547	5	744
7	968	6	80
8	7	7	386
9	9	8	18
12	206	9	13
15	3563	10	2
20	3	11	359
21	3722	12	45
22	918	13	794
23	5019	14	2
24	9	15	174
25	30	16	16
29	812811	17	145
31	4	18	15
32	20	19	143
33	12	20	2
34	893	21	154
35	14	22	12
36	242149	23	860
39	3007	24	2
Total	1074974	25	22
		26	308
		27	1049
		28	360
		29	47879
		30	13
		31	2
		32	1002
		33	1
		35	9
		36	20332
		37	2
		38	9
		39	936
		40	4
		Total	77216

Figure 4. Training and Testing Counts per Class Post Unique Removal

4. Experimental Evaluation

4.1. Dataset Preprocessing

We took note of [11], [3], and [12] and removed duplicate entries from the training set and the test set. This was also done in the interest of overall computation time. It is possible that entries between the test and training sets correspond, however within each respective dataset there are no identical entries. We reduced the training set from 4898431 entries to 1074974 entries. The test set was reduced from 311029 entries to 77216 entries. We noticed that two of the classes within the training set accounted for a disproportionate amount of the data, so we down sampled these classes by 100 times. Classes with fewer than 20 samples we removed entirely from the training set. In order to format the data for learning we normalized each vector element to a 0-1 range using a linear min-max scaling across all data on an element-wise basis. For the categorical data, we assigned integer values corresponding to an index in a number of categories prior to conducting the scaling. We hypothesize that a different encoding could yield superior results e.g., a one-hot encoding. However, this increases the feature vector length along with computation time and is orthogonal to the analysis in this paper. We trained both the RBF SVM and the W-SVM using these chosen parameters. We used the 1 vs. Rest W-SVM when comparing against the 1 vs. 1 RBF SVM – the default scikit-learn SVM. In order to determine a good value for C and γ in our experiments, we first performed a 3-fold cross validated order of magnitude grid search over the training set. We used a grid of $C, \gamma \in \{10^{-5}, 10^{-3}, \dots, 10^5\}$. Based upon accuracy over this grid search, we selected a C value of 1000 and a γ value of 0.1 for training our classifiers.

4.2. Closed Set Protocol Evaluation

We evaluated classification error in terms of both closed and open set accuracy. For the closed set protocol this is simply the number of correctly classified test instances out of the total number of test instances, which does not account for unknown classes: i.e., all test instances with labels not present in the training set will be misclassified under this protocol. For the RBF SVM we obtained a closed set accuracy of 91.1%. For the W-SVM we obtained a closed set accuracy of 90.1%. It is not surprising under a closed set regime that the RBF SVM outperforms the W-SVM.

Despite the fact that out of the classes in the test set there were 25 classes that were not seen in training, due to disproportionate sampling that we did not account for, only about 5% of the test data consists of unknown classes. This is one of the reasons for the relatively high accuracy numbers (i.e., >90%) even under the closed set regime. We also found one label in the training set that was not found in the test set, which could contribute to misclassification error in test. Overall, the number of classes in training was 14 and the number of classes in testing was 38. There were 25

classes found in the testing set that were not in the training set.

4.3. Open Set Protocol Evaluation

For evaluating under an open set protocol we thresholded probabilities returned by both W-SVM and the Platt-calibration on the RBF SVM by a variety of different thresholds. We do not address threshold selection in this paper, although this has been discussed in several other works including [7] and [6]. Instead, we inspected results under a variety of thresholds from 0.1 to 0.3. For 0.1 the W-SVM scored an accuracy of 91.3% and for 0.3 scored 90.8%. For 0.1 the RBF SVM scored 91.1% and for 0.3 it scored 91.5%. This suggests that generally, even for an open set protocol on the testing set, while using an open set classifier closes the accuracy gap, the thresholded RBF SVM generally achieves slightly superior classification performance.

This discrepancy is due to the fact that despite having a large number of unknown classes, the majority of the samples in the test partition are still from the common 13 classes seen in the training set. With a large proportion of these corresponding to “normal” behavior. Thus, although the KDD’99 dataset has open set characteristics, it is still mostly closed set in nature. However, when we independently evaluate open set performance on known and unknown classes within the test set, we find a telling insight: namely, for only known classes the classification performance of the RBF SVM is 96.2% and the W-SVM is 95.1% under threshold 0. For the RBF SVM it is 96.2% under 0.1 and the W-SVM is 94.6%. Under a threshold value of 0.3, the RBF SVM exhibits accuracy of 96.2% and W-SVM of 93.7%. This suggests that RBF SVM predictions are highly confident. On the other hand, the W-SVM suffers performance decrease in the purely closed set regime when classifying sparse instances. However, for the unknown samples in the test set we see a much different trend. For the unknown instances in the testing set, for RBF SVM at a threshold of 0.1 no samples are rejected and at a threshold of 0.3 only 6% of the samples are rejected (correctly classified as unknown). For the open set classifier on the other hand, at a threshold of 0.1, 30% of the unknown samples are correctly rejected. At a threshold of 0.3, 39% of the samples are correctly rejected. Thus, the open set classifiers serve better to reject unknowns, although there does appear to be significant data overlap between known and unknown classes due to the fact that a minority of the unknown samples are rejected. This could be due to an insufficiently expressed feature space. Perhaps a one-hot vectorization of categorical data would be better for future work.

4.4. Applying Open Set Classifiers: Quantifying Cost of Unknown

An important question to address is what is the number of unknown samples that one would expect to see in a real network traffic environment? This is not simple to ascertain; also, even quantifying “unknown” is difficult because

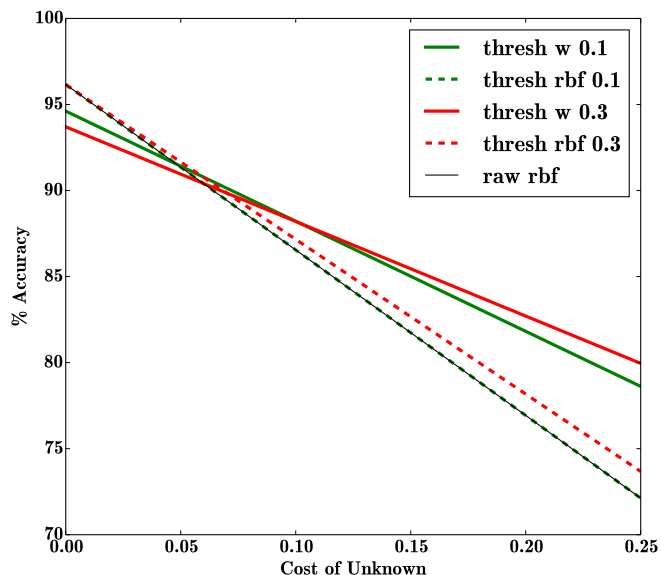


Figure 5. This figure depicts decay in accuracy as we up-weight the cost of labeling an unknown instance as coming from a known class for both open-set and closed-set classifiers. When attributing no cost to making classification errors on an unknown class depicts, an unthresholded RBF SVM is superior to the W-SVM. The situation quickly changes, however, as we upweight the cost of unknowns. This is because, for an open set classifier with thresholds 0.1 and 0.3, unknowns are better rejected (0.1: 30% and 0.3: 39%) than with a closed set classifier (0.1: 0% and 0.3: 6%).

different classes of data are often ill-defined – e.g., *should* a previously unseen form of novel “normal” behavior be labeled as unknown. According to our open set protocol the answer is no, because we only have labels of finite granularity, but in a realistic operational setting, the story may be different.

In Sec. 4.3, we saw that the closed set classifier performed comparably to the open set classifier on the KDD’99 dataset open set protocol, when considering both known and unknown classes, closing the performance gap in the purely closed set regime. However, it is important to consider that the majority of the KDD’99 test partition is dominated with “known” benign samples, which may well be anomalous with respect to the support of the training data. With that in mind, amidst a sea of non-malicious traffic are anomalies that can compromise the secure state of the network. Thus, the *cost* of mislabeling an unknown sample is highly dependent on the operational constraints.

In Fig. 5, we evaluate the impact on perceived accuracy with respect to number of unknown samples, or equivalently, “cost of unknown”. Weighting the relative importance of accuracy on known and unknown partitions of the test data. When accuracy on unknown samples is given no weight, the RBF-SVM dominates, but as we slightly upweight the cost of unknown to just 7%, (i.e., accurate classification of

unknown is given 7% weight, whereas accurate classification of known is given 93%) the W-SVM's performance quickly eclipses that of the RBF SVM, with an increasing performance gap as cost of unknown increases.

Thus, open set recognition has tremendous potential in the intrusion recognition realm depending on the application. Curves like in Fig. 5 can serve to choose between open-set and closed-set classifier for the application at hand.

5. Discussion

Within our training set we had an extra label (34) that was not present at test time for our fine-grained experimental protocol. This likely contributed to misclassifications, resulting in reduced accuracy. Also, while we performed rebalancing on our training set, in terms of reducing over-represented samples, the only processing that we did on our test set was to remove duplicates. Thus, this also likely reduced classification accuracy because the training set bias no longer matched the test set bias.

To thoroughly evaluate closed-vs-open set classification within the intrusion and malware recognition domain, more datasets are required, our experiments have offered the first experimental proof points that suggest that using an open-set approach open has previously untapped potential for distinguishing novel types of behaviors. For the purposes of malware classification, this problem space had yet to be applied in an experimental evaluation until now.

The KDD'99 dataset is ripe with flaws and, as we discussed in the previous section is still very closed. This presents limitations with the approach and the degree to which we can extrapolate from our results to real intrusion detection settings. A more realistic dataset would allow for more thorough evaluation of our open set protocol. Moreover, operational considerations, including the cost of unknown with respect to a particular system and the degree of troubleshooting expertise of system/network operators plays an important role when considering system design. As our experiments show, assuming low "cost of unknown", running experiments on the raw KDD'99 dataset, open set recognition provides little benefit due to the closed nature of the dataset; we have to open the dataset to get serious benefit. The degree to which open set recognition works well is largely a function of the dataset. KDD'99 is extremely closed, because it was designed as a closed set benchmark. This is why we had to, contrary to the conventional KDD'99 protocol, perform evaluation on fine-grained individual attack types, not just attack meta-types (cf. Fig. 2), for which the KDD'99 dataset is not closed set, there is likely noticeable overlap between malware categories, so the KDD'99 dataset may be more closed than our open set protocol suggests. Despite this, we were still able to demonstrate performance gains offered by an open set classifier.

Overall, the RBF SVM outperformed the W-SVM on a purely closed set evaluation of the data, but we saw a shift in applying an increasingly open set evaluation where the W-SVM took the lead in performance accuracy. In the future, these techniques should be extended to a better dataset and

therein re-evaluated. We demonstrated that When the cost of unknown increases, however, by applying an open set protocol, we were able to garner useful results that closed set classifiers cannot deliver. A similar protocol can easily be applied to a more realistic dataset in the future.

6. Acknowledgements

This research project was funded by NSF award number 1359275.

References

- [1] E. Rudd, A. Rozsa, M. Gunther, and T. Boulton, "A survey of stealth malware: Attacks, mitigation measures, and steps toward autonomous open world solutions," *arXiv preprint arXiv:1603.06028*, 2016.
- [2] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.
- [3] M. Tavallaei, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.
- [4] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.
- [5] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 1999, pp. 120–132.
- [6] W. J. Scheirer, L. P. Jain, and T. E. Boulton, "Probability models for open set recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 11, pp. 2317–2324, 2014.
- [7] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boulton, "Toward open set recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1757–1772, July 2013.
- [8] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92. New York, NY, USA: ACM, 1992, pp. 144–152. [Online]. Available: <http://doi.acm.org/10.1145/130385.130401>
- [9] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, 2006.
- [10] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *ADVANCES IN LARGE MARGIN CLASSIFIERS*. MIT Press, 1999, pp. 61–74.
- [11] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, 2001, pp. 5–8.
- [12] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38*, ser. ACSC '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2005, pp. 333–342. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1082161.1082198>

Improving Partial Fingerprint Recognition

Costas, Andrea
ayc2@rice.edu
Rice University

Boult, Terrance
tboult@vast.uccs.edu
University of Colorado at Colorado Springs

Abstract—In today’s world, biometrics are becoming more and more popular for the levels of security and convenience they provide to users. Apple has been including fingerprint scanners on their most recent iPhones, and a variety of other big names in technology are moving in the same direction. With the increased use of fingerprints, the need to have better and smaller scanners is increasing. With that need comes an urgency to develop forms of authentication using smaller amounts of information. The ability to use partial fingerprints effectively would be incredibly beneficial moving forward. This approach would require finding a balance between user security and program efficiency. This work proposes an approach to finding that balance through the use of SIFT features.

Index Terms—fingerprints, partial fingerprints, SIFT features, fingerprint analysis

I. INTRODUCTION

The rise² of biometrics as a means of security has been a long time coming. It has helped redefine user authentication procedures by antiquating the use of passwords and PIN numbers. Passwords and PINs can be stolen with relative ease, but fingerprints on the other hand are unique and fairly difficult to fake. Fingerprints provide added security with the promise of making sure no one can access an account or device that does not belong to them. Fingerprinting is far from a new concept, but one that has encountered and continues to encounter many challenges throughout the course of its development.

Maltoni et al. [1] pointed out that the adoption rate of biometrics has been slower than what was expected due to a general lack of awareness of its potential. For a long time, the use of biometrics has been greeted with hesitation. This has all been changing over the past few years now that cell phone manufacturers are placing fingerprint scanners in everyone’s hands. The recent incorporation of fingerprint verification systems in cell phones has eased a lot of public concern leading to an increase in the popularity of biometrics in the commercial world. In this commercial world, the ability to reduce production costs is valuable, and one way to do that is by creating smaller products. So the ability to effectively use smaller scanners escalates in value. Smaller scanners

¹ A. Costas is participating in a Research Experience for Undergraduates (REU) with the Department of Computer Science, University of Colorado Colorado Springs, CO, 80918 USE under NSF Award No. 1359275

²<http://www.csoonline.com/article/2891475/identity-access/biometric-security-is-on-the-rise.html>

³The term “partial fingerprints” is used to mean small fingerprint regions as technically speaking, all fingerprints are partial fingerprints.

⁴<http://zwipe.com/news/zwipe-introduces-genuine-hid-technology-biometric-cards/>

mean smaller fingerprint regions which increases the need for programs that can authenticate users working with relatively small pieces of information while also ensuring that other users are not incorrectly authenticated.

There exist fingerprint matching algorithms already, however, none of them has proved very effective working with small prints. This work consists of testing current, widely-used matching procedures and then some that are not as widely-used to see what proves most effective and promising.

II. PROBLEM DEFINITION

Improving the means of working with partial fingerprints would have significant implications for the uses of biometrics, more specifically, the uses of fingerprints in commerce. There have already been applications of partials in consumer products such as cell phones, but there is potential for expansion into other products and fields. Fingerprint protected credit cards, USB drives, and more are all feasible through effective uses of partial fingerprints. But in order to use them, there must be ways to confidently authenticate them. Therefore, current authentication procedures must be assessed for their effectiveness, and then a new method may be necessary in order to see any actual improvement.

III. PREVIOUS WORK

There has been plenty of research conducted in the past on fingerprints and fingerprint matching methods and their challenges. However, most of these methods are unsuitable when considering the properties (or lack thereof) of partial fingerprints. Many of the key features within fingerprints are lacking when working only with a small portion, which forces the development of a new field of thought when studying such prints. However, there is still not a lot of information regarding the best way to approach this.

A. Sensor Sizes

Mainquet et al. [2] studied sweep sensors and reasonable minimal sizes for those sensors. They used minutia-based identification software and found that 7mm was the minimum width for a sweep-type sensor to perform with acceptable accuracy. It is important to note that sweep sensor prints are quite different from partial fingerprints as they can often result with fairly full-sized images. These sensors essentially take multiple partial images and arrange them together to create a full, mosaic-like image.

B. Single-Chip Sensor and Identifier

Shigematsu et al. [3] researched a chip architecture that was comprised of both a fingerprint sensor and a fingerprint identifier. The identifier works with an array of pixels that are processed in parallel, so each fingerprint is handled and studied pixel by pixel. Once the print was passed through the sensor, the chip would store the data of an initial print scan. Later, when a new finger was placed on the chip, it would generate an image of the fingerprint by sensing the shape of it. It proceeded to binarize that image and shift it around to allow for various finger positions. Finally, it would compare the two prints and report its result. In testing, the chip worked correctly 99% of the time. This study focused more on creating smaller scale fingerprint sensors and did not test how effective it was when working with partial prints but the rapid processing it provides makes it an intriguing option to look into in the future.

C. Multi-pass Matching Algorithm

Jea et al. [4] presented a multi-pass partial fingerprint matching algorithm that overcomes many of the typical challenges partials provide such as rotations and distortions by using localized features. Their algorithm is based on triangular matching and secondary features/ minutia points. It accounts for distortions of minutia and different orientations that could be caused by differences in pressure during the fingerprinting process. In many matching algorithms, many matching algorithms work by aligning two fingerprints and finding a direct correspondence between minutia points. In this algorithm, however, does not require alignment. It basically tries to find local matches, and then expand onto a more global-scale to try to conclude if the prints matched. This algorithm proved effective and provided tolerance of a large number of distortions and effective use of secondary and localized features.

This all sounds great and useful, however, the smallest size of prints they worked with were a 60% of the original full sized print. According to their graph, this means that they had anywhere from 25 to 55 minutia points, so the partials were still of significant size and much larger than the partials generated in section V-A.

D. Region of Interest Minutia Matching

Bhargava et al. [5] studied the limitations of image processing, particularly fingerprint processing, when it comes to image quality. They present a solution to this problem by choosing a Region Of Interest (ROI). ROI is essentially a segmentation procedure that helps represent the image in a simpler way. It can make analysis easier by making the image appear more meaningful. The minutia within the ROI is first marked, then the locations of those minutia points are compared for verification. This provides an interesting idea for if a partial can be considered an ROI and matched as such, however ROI picks the best possible, most representative region, and partials are not necessarily representative at all. University of Colorado, Colorado Springs

E. Minutia Recognition

There have been a variety of different studies along the lines of partial fingerprint recognition, but in this one in particular, Jea and Govindaraju [6] conducted research on a minutia-based partial fingerprint recognition system. They developed a system that uses localized secondary features of minutia and made use of a neural network. They acknowledged that a brute force approach to this problem would be unfeasible due to how many possibilities the program would have to consider. They also discarded approaches that make use of global features seeing as how partial fingerprints can often have no global features at all. The neural network they developed was based on a system of similarity scores and showed that the accuracy of the fingerprint matching improved when working with images larger than $0.32'' \times 0.46''$. This size is approximately 60% of a full-sized fingerprint. When the prints were smaller than that, the performance dropped dramatically.

F. Fingerprint Recognition Using Robust Local Features

Mishra et al. [19] pointed out that there already exist recognition techniques for fingerprints and that most rely on minutia matching methods that are not rotation-invariant. For this reason, they often fail with transformed prints and partials. They tested SIFT's performance matching prints and found that SIFT was effective in matching and feature extraction, and although they tested on smaller prints, they did not test on anything similar to the sizes seen in this paper.

G. SIFT Fingerprint Identification

Zhou et al. [20] discussed that the original SIFT algorithm would not be suitable for fingerprint identification due to the similar patterns of fingerprint ridges. They proposed a minutia descriptor based on SIFT in hopes of improving how quickly prints can be verified. They had good results not only working with regular prints but also cracked and low quality prints as well.

IV. PROPOSED SOLUTION

The proposed solution is comprised of a few major steps. The first pertains to acquiring the data that is to be used for training and, eventually, testing. The second has to do with creating a baseline of sorts. This means testing the performance of pre-existing matching algorithms in order to have a comparison for performance later. The final step is dependent on the previous step. If the matching algorithms perform well, then machine learning can be used to create a program that automatically chooses the best matching algorithm for a task depending on the fingerprint. If the matching algorithms do not perform well, then machine learning will be used to create a new, more effective way of matching partial prints to their full-sized counterparts. The data created in the first step will be used to train a computer to identify matches and non-matches.

A. Data Collection

This step is primarily necessary for the training process. Full sized fingerprints from the 2000, 2002, and 2004 Fingerprint Verification Competitions (FVC) [1] will be the source of data for this research. Each year has approximately 3,500 different full-sized fingerprints. All of them will be used. These full-sized prints will be partitioned into a variety of smaller images of a size that would resemble the partial prints acquired by a small sensor. In this case, three different sizes are being generated. The largest size is 192 by 192 pixels. This is the size on MasterCard and Zwipe's new biometric credit cards³. There are also sizes of 128 x 128 and 96 x 96. Although these sizes are not regularly used, they would provide a useful comparison to possibly gauging how small is too small. The full-sized fingerprints will also be rotated and then partitioned in order to provide data on different transformations of prints that can occur. These prints will be used later to train a deep neural network to either choose which is the best matching algorithm to use, or to create a new network that learns when prints match and when they do not. Only about 80% of the generated partials will actually be used in training. The rest will be reserved for testing.

B. Base Line

This steps consists of creating a baseline or, in other words, testing the performance of existing matching algorithms on generated partial prints. A few different matching algorithms will be tested. Minutia Cylinder Code (MCC) [7], Protected Minutia Cylinder Code (PMCC) [8], and Bozorth3 [9] which is the NIST Biometric Image Software (NBIS) matching algorithm. These algorithms have proved as effective on full sized prints, however, they have never been tested on a smaller scale.

1) *Minutia Cylinder Code*: Capelli et al. [7] introduced the Minutia Cylinder Code (MCC) in a paper aiming to create a fingerprint matching algorithm that would focus solely on local minutia matching and combine the advantages of neighbor-based structures and fixed-radius structures while cutting out the drawbacks of each. The neighbor-based approach focuses on the K spatially closest neighbors while focusing on some central minutia point. This method is tolerant of sparse and missing minutia points. It comes with a couple of drawbacks such as sophisticated local matching and problems with the handling of radial angles. The fixed-radius approach can also lead to mismatch die to local distortions or location inaccuracy. The basic idea behind MCC is that a local structure is assigned to each minutia. These structures represent spatial and directional relationships between that minutia and those in its surrounding neighborhood. These parameters end up being represented by a cylinder where the base and height correspond to each parameter.

2) *Protected Minutia Cylinder Code*: Protected Minutia Cylinder Code (PMCC) is based off of the MCC algorithm but its main purpose is to keep minutia templates from being acquired from PMCC templates. It still uses the same system

of cylinders to work, however, each cylinder is transformed permanently in order to provide that "protection" it promises.

3) *Bozorth3*: Bozorth3 is NBIS's minutia matching algorithm [9][10][11]. Essentially, the algorithm computes relative measurements from each minutia and builds comparison and compatibility tables that can combine clusters to calculate a match score. The higher this score is, the more likely it is that fingerprints in question came from the same person.

C. Learning and Moving Forward

The final part depends on the previous part. If the matching algorithms perform well, then the next step would be to use machine learning to learn how to choose the best matching algorithm for a given task. If the matching algorithm performs poorly, this step could consist of searching for other methods of matching prints and analyzing them or creating a deep neural network and training it to classify partial fingerprints as matches or non-matches to a full sized fingerprint.

Some of the other alternatives to traditional fingerprint matching would be using and testing feature descriptors for their performance on prints. Using SIFT would be a good possibility simply because it has never been tested on prints as small as the ones studied in this paper. Other papers have achieved good performance using SIFT descriptors and matching on large prints.

V. PRELIMINARY RESULTS

A. Data Generation

Each image in the 2000, 2002, and 2004 FVC databases was passed through an algorithm that simply, after traversing every few pixels, an image was generated that was 96 x 96 pixels, another that was 128 x 128, and another that was 192 x 192. These sizes represent different portions of a full-sized print. The 192 x 192 is about $\frac{1}{4}$ of a full-sized print. The 128 x 128 is about $\frac{1}{8}$ and the 96 x 96 is just a bit bigger than $\frac{1}{16}$ of a full print. To provide some context, an Apple TouchID scanner uses prints that are 160 x 160 pixels which means they are approximately $\frac{1}{6}$ of a standard full print.

Not all of the generated partials were kept. An analysis was done to examine the average contrast over each generated image. This was done to get an estimate as to how much white space there was in each new partial. If an image had too much white space, it was discarded. See image . Such images basically did not have a large enough print portion on it for them to be worth keeping. They would not provide substantial information, so they would be essentially useless in the future when it came to testing and training. See Table 1 for more information on how many images were kept and how many were discarded on average.

B. Feature Extraction

In order to get the matching algorithms discussed in part IV-B working, they must be provided with a list of the minutia in each of the prints to be matched. Regularly, a full-sized fingerprint could have around 100 minutia points,

FINAL, AUGUST 2016

TABLE I

AVERAGE NUMBER OF IMAGES GENERATED AND KEPT FOR A SAMPLING OF FINGERPRINTS

Images kept with image size and shift size (5)			
	96 x 96	128 x 128	192 x 192
Generated	5616	4615	3016
Kept	3717	3305	2481

but with the smaller sizes in the generated data set, there are not nearly that many minutia points. So generally, fingerprint images must be passed through some kind of minutia extraction algorithm. The minutia extractor originally chosen was DigitalPersona's FingerJetFx. When trying to extract from the generated images, the FingerJetFX extractor failed completely. It worked excellently on full sized images, but upon trying it on the partials, it refused to run. It would not run even on the largest of the partial sizes. Another minutia extractor was tried, this time it was NBIS MindTCT extractor. The MindTCT was able to extract minutia. But the failure of the FingerJetFX raised a red flag.

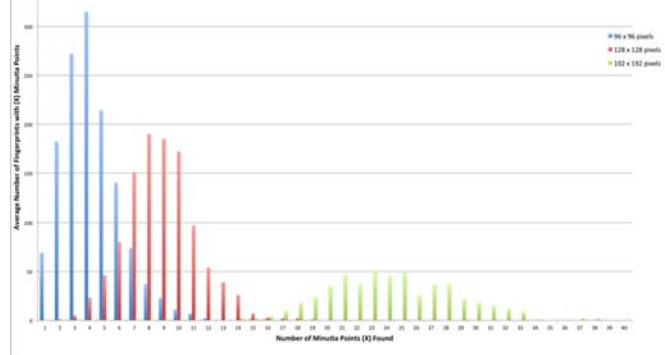
The matching algorithms most likely had a minimum number of minutia required to be able to try to match two prints. If the partials had less minutia than the lower bounds of the matching algorithms, then it would be impossible to test the effectiveness of those algorithms in the first place. Or rather, the algorithms would immediately be deemed to be ineffective on small prints. Looking into it further, the MCC is programmed to only run if there are at least 4 minutia points available. The PMCC requires at least 10 as does the Bozorth3 algorithm.

With this information on the limits of the matching algorithms, it was time to find out how much minutia could be extracted on average from the different sized of partial prints. Figure 1 shows the relationship between how many minutia points were found on average for partial fingerprints. The partials of size 96 x 96, generally found between 1 and 5 minutia points per image. The 128 x 128 found about around 6 to 11 minutia points and the 192 x 192 found anywhere from 20 to 28 minutia points.

When considering how many minutia points were extracted and the limitations of the different matching algorithms, it seems that creating a neural network to choose the best matching algorithm for a specific purpose may be, essentially, useless. None of the matching algorithms previously discussed would be able to process most of the 96 x 96 partials. The algorithms might be able to process some of the 128 x 128 partials, but only the MCC would really be able to work with most of them. This means that only the 192 x 192 holds good possibilities to be matched using the different matchers discussed.

The question now becomes what can be done to improve the matching processes of the smallest fingerprints in the database. University of Colorado, Colorado Springs

Fig. 1. Relationship between fingerprint sizes and how many minutia points were extracted.



C. SIFT Features

Following down the path of some other researchers, testing SIFT seemed promising. Before matching, partial prints were put through a contrast normalization algorithm and they were also blurred. This was done to remove any sort of randomness and inconsistencies between different print scans.

Here a couple of prints were compared with partials generated from seven different scans of the same finger. In other words, if finger A was scanned eight times, then finger A-1 is compared with the generated partials of finger A-2 through finger A-8. Due to limited times, this was only tested on 2 different fingers, but moving forward with this project, there will be many more sets of prints tested. But here are the results for how matching went with different sizes of partials.

In the first experiment, the threshold was set to 10. This means that in matching, there had to be at least ten good matches (where "good" is defined as matches that perform well under Lowe's ratio test []). If there were less matches than that, then there would not be a conclusive match. See Figure 2. Not surprisingly, the size 192 partials matched best here by matching just over 30 percent of the time. Yet, the results leave a lot to be desired.

The next experiment tested a lower threshold of 5 in hopes of getting improved performance on the matches. While the results improved, it was still not enough to be considered satisfactory. On average, the size 192 was accurate a little more than 50% of the time.

The threshold was lowered once more to 1. Only one match was necessary in order to match a partial to a full sized print. See Figure 3. Finally, there were better results.

It seemed concerning to lower the threshold so much, considering that one match is seemingly nothing to prove any degree of accuracy or security. However, a couple of brief preliminary tests were conducted with non-matching pairs. On both of the tests, the threshold was set to 1, the lowest of all the thresholds previously tested for true matches. When the results came back, all of them came back negative. That is to say that no prints or partial prints from different fingers matched at all. This is promising for the continued testing of this method, but not enough to prove anything definitively.

Prints will continue to be tested to be sure that this method does not pose any serious security risks for users.

Fig. 2. Percentage of partial prints accurately matched per partial size while under a threshold of 10.

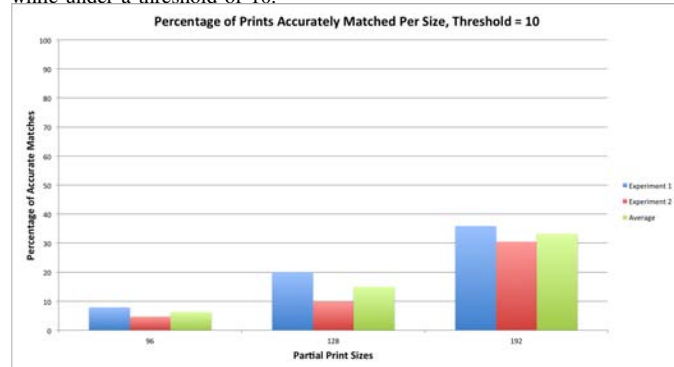


Fig. 3. Percentage of partial prints accurately matched per partial size while under a threshold of 5.

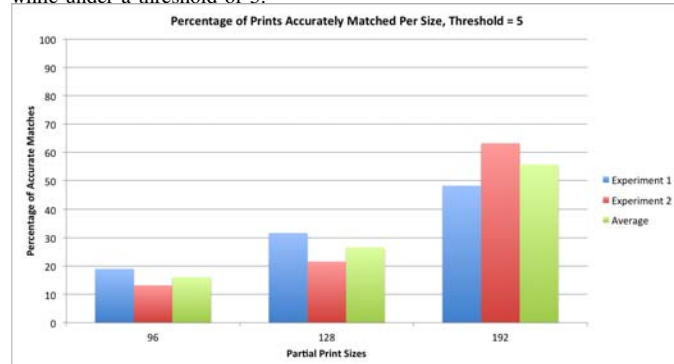
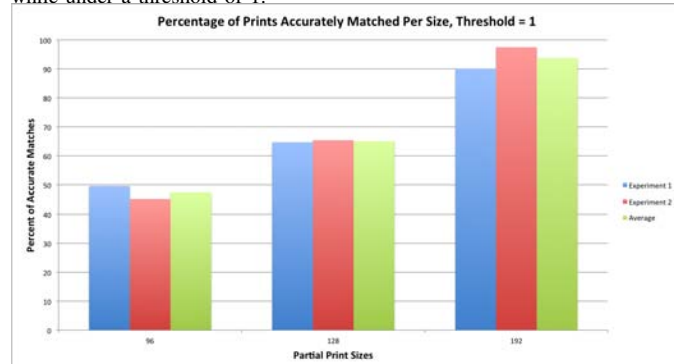


Fig. 4. Percentage of partial prints accurately matched per partial size while under a threshold of 1.



VI. CONCLUSION

It is far too soon to have any conclusions other than some educated guesses as to which path to take moving forward with this research. Definitely, using only minutia based matching methods would not be effective for such small prints, but that does not mean that they cannot serve some other purpose in the future.

VII. FUTURE WORK AND IMPROVEMENTS

Work is far from over. Despite some challenges with testing, the use of SIFT has proved promising for authenticating fingerprints so it seems that the next step should be to try other robust feature transforms. Aside from Scale-Invariant Feature Transform (SIFT) [21], there is also Speeded Up Robust Features (SURF) [18], Binary Robust Invariant Scalable Keypoints (BRISK) [16], KAZE, and Fast Retina Keypoint (FREAK) [22]. All of these hold potential for improving partial fingerprint recognition.

Leutenegger et al. [16] proposed BRISK (Binary Robust Invariant Scalable Keypoints), another method for feature detection and matching. It primarily uses brightness comparisons to form a binary description. It compares well with SIFT and SURF, so it will be worth looking into in the future, but if it relies solely on the binary description, it may not be suitable for fingerprint matching simply because the prints are already represented in a binary fashion.

Proceeding on-wards, after testing other feature transforms, it may be interesting to combine a few methods. This could mean either combining a couple of feature transforms or combining a feature transform with minutia data or both. Now minutia points and keypoints have been used together in the past, but not at a scale as small as this one, which leaves the door open for more research. In the future, the use of a neural network for this will be considered as well. The hopes are to see more improvement in the matching of the smaller prints like the 96 x 96 and the 128 x 128.

REFERENCES

- [1] D. Maltoni, D. Maio, and A. K. Jain, Handbook of fingerprint recognition [With DVD ROM]. United Kingdom: Springer-Verlag New York, 2009.
- [2] J.-F. Mainguet, W. Gong, and A. Wang, "Reducing silicon fingerprint sensor area," in Biometric Authentication, Springer, 2004.
- [3] S. Shigematsu, H. Morimura, Y. Tanabe, T. Adachi, and K. Machida, "A single-chip fingerprint sensor and identifier," Solid-State Circuits, IEEE Journal of, vol. 34, no. 12, 1999.
- [4] T.-Y. Jea, V. S. Chavan, J. K. Schneider, and V. Govindaraju, "Partial Fingerprint Matching."
- [5] "Region of interest minutia matching - Google Search." [Online]. Available: <https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=region%20of%20interest%20minutia%20matching>. [Accessed: 08-Jul-2016].
- [6] T.-Y. Jea and V. Govindaraju, "A minutia-based partial fingerprint recognition system," Pattern Recognition, vol. 38, no. 10, 2005.
- [7] R. Cappelli, M. Ferrara and D. Maltoni, "Minutia Cylinder-Code: a new representation and matching technique for fingerprint recognition", IEEE Transactions on Pattern Analysis Machine Intelligence, vol.32, no.12, December 2010.
- [8] A. Rozsa, A. E. Glock, T. E. Boulton, "Genetic Algorithm Attack on Minutiae-Based Fingerprint Authentication and Protected Template Fingerprint Systems", CVPR2015.
- [9] R. Vaan, "NIST MINDTCT and Bozorth3 Review — SourceAFIS."
- [10] "Biometric System Laboratory." [Online]. Available: <http://biolab.csr.unibo.it/research.asp?organize=Activities&select=&selObj=81&pathSubj=111%7C%7C8%7C%7C81&Req=&>. [Accessed: 03-Jun-2016].
- [11] N. US Department of Commerce, "NBIS." [Online]. Available: <http://www.nist.gov/itl/iad/ig/nbis.cfm>. [Accessed: 10-Jun-2016].
- [12] A. K. Jain, J. Feng, and K. Nandakumar, "Fingerprint Matching," Computer, vol. 43, no. 2, 2010.
- [13] G. Hinton, "Deep belief networks," Scholarpedia, vol. 4, no. 5, p. 5947, 2009.

- [14] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: <http://cs231n.github.io/convolutional-networks/>. [Accessed: 10-Jun-2016].
- [15] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR05), 2005, vol. 1.
- [16] S. Leutenegger, M. Chli, and R. Y. Siegwart, BRISK: Binary robust invariant scalable keypoints, in 2011 International conference on computer vision, 2011.
- [17] F. Alcantarilla, A. Bartoli, and A. J. Davison, KAZE features, in European Conference on Computer Vision, 2012.
- [18] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, Speeded-up robust features (SURF), *Computer vision and image understanding*, vol. 110, no. 3, 2008.
- [19] R. Mishra and R. Mishra, Fingerprint recognition using robust local features, *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 6, 2012.
- [20] R. Zhou, D. Zhong, and J. Han, Fingerprint Identification Using SIFT-Based Minutia Descriptors and Improved All Descriptor-Pair Matching, *Sensors (Basel)*, vol. 13, no. 3, Mar. 2013.
- [21] T. Lindeberg, Scale invariant feature transform, *Scholarpedia*, vol. 7, no. 5, 2012.
- [22] A. Alahi, R. Ortiz, and P. Vandergheynst, Freak: Fast retina keypoint, in *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, 2012.

Assessing Threat of Adversarial Examples on Deep Neural Networks

Abigail Graese Andras Rozsa Terrance E Boulton
 University of Colorado Colorado Springs
 agraese@uccs.edu {arozsa,tboulton}@vast.uccs.edu

Abstract—Deep neural networks are facing a potential security threat due to the discovery of adversarial examples, examples which look normal but cause an incorrect classification by the deep neural network. For example, the proposed threat could result in hand-written digits on check or mail being incorrectly classified but looking normal when humans see them resulting in mail being sent to a destination chosen by the adversary. This research assesses the extent to which adversarial examples are a major security threat when combined with the normal image acquisition process. This process is mimicked by adding small transformations that could be the result of acquiring the image in a real world application, such as gathering information for use by an autonomous car with a camera or using a scanner to acquire digits for a check amount. These small transformations negate the effect of a large amount of the perturbations included in adversarial examples, causing a correct classification by the deep neural network, therefore decreasing the potential impact of the proposed security threat. We also show that the already widely used process of averaging over multiple crops neutralizes most adversarial examples.

I. INTRODUCTION

The solving of classification problems in machine learning has recently made significant progress through the use of deep neural networks, or deep learning [7, 8, 11]. Deep neural networks (DNNs) require supervised learning, which is the use of a training set that contains known outputs for the inputs during the training of the DNN. After training is completed, when presented with unknown inputs, the DNN is able to classify the inputs with exceptional accuracy.

Although DNNs have a high accuracy rate, images known as adversarial examples trick the DNN into classifying an image incorrectly despite humans seeing almost no difference between the original and adversarial image. Incorrect classification occurs close to 100% of the time when used as direct inputs to the DNN the adversarial example was created for. Recently, researchers have proposed that adversarial examples can “seriously undermine the security of the system supported by the DNN,” [3] because the incorrect classification could potentially lead to an incorrect action with consequences. For example, if a stop sign was crafted as an adversarial example, an autonomous vehicle could complete an incorrect classification of the sign and cause an accident to occur [12].

In real world applications of deep learning however, the input to a DNN will be coming from an outside source, such as a picture from a camera or a scanned image. The input

This work was supported by the 2016 Research Experience for Undergraduates (REU) program at the University of Colorado Colorado Springs (NSF Award No. 1359275).

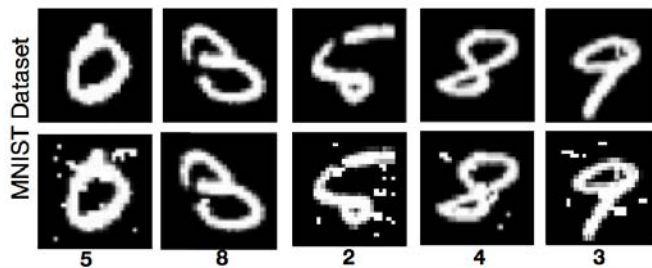


Fig. 1: **Authentic Examples versus Adversarial Examples** The images on the top row of are legitimate images. The images on the bottom row of are adversarial examples, and the numbers below each of those image is the number that the DNN mistakenly classifies the adversarial examples. Adapted from [3].

images will always contain slight transformations, such as shifting or blurring, and perturbations, such as noise, due to the imperfect capture of the input, which perturbs the input to the neural network from the intended input slightly.












The validity of a situation where an adversarial example could be crafted into a real world application input which then survives the image acquisition process needs to be assessed. This research assesses the extent to which adversarial examples are handled and classified correctly, simply through the natural process of acquiring the image, which renders the input nonadversarial. The acquisition process is mimicked in this paper by performing small transformations that could be expected in normal image acquisition.

We also note that all state of the art deep convolutional neural networks (DCNNs) use multiple crops and often multiple networks in reaching their final decision. To date no paper on adversarial examples has examined if they survived this widely used component of DCNNs. We show that even with only 5 crops (on non-transformed), compared to the 10s to hundreds used in state of the art networks, the majority of adversarial images will be correctly classified.

By assessing the validity of the extent to which adversarial examples are a security threat, future applications using DNNs will be more informed about the extent and effect of potential security risks facing the networks.

II. RELATED WORK

Deep neural networks are learning models that produce state-of-the-art results for several types of classification and recognition problems [7,8]. Szegedy et al. [4] discovered that

 (a)	 0.4860 549 26 (b)	 0.3885 1070 51 (c)	 0.3478 1333 64 (d)	 0.3132 1577 76 (e)	 0.1949 2603 127 (f)
	 0.7727 358 57 (g)	 0.6527 666 113 (h)	 0.5665 968 170 (i)	 0.4919 1252 226 (j)	 0.4258 1514 252 (k)

(a)

Fig. 2: **FGS Adversarial versus FGV Adversarial** The metrics underneath the numbers are the PASS, L_2 norm and L_∞ norm respectively. Image (a) in each row of the figure is the original MNIST image. Images (b)-(f) are FGS adversarial examples. Image (b) has the minimum perturbation and ϵ required to create an adversarial. Image (c), (d), (e) and (f) have an ϵ of 0.20, 0.25, 0.30, and 0.50 respectively. For MNIST, the data was scaled to (0,2), so an ϵ of 0.2 means binary sign image is effectively scaled by 20%, i.e. 51 gray values. Human perception can see deviations of 5-10 gray values when doing a comparison. Images (g)-(k) are FGV adversarial examples. Image (g) has the minimum perturbation required to create an adversarial. Image (h)-(k) have 2, 3, 4, and 5 times the minimum perturbations respectively.

there exist perturbations which when included in an image cause an incorrect classification by the DNN but which are “imperceptible to humans; these examples were classified as “adversarial examples.”

Since this discovery, several advancements in the understanding and creation of adversarial examples have taken place. Sabour et al. [16] demonstrated that the existence of adversarial examples could be the result of the architecture of DNNs themselves. Goodfellow et al. [5] presented the fast gradient sign (FGS) method for generating adversarial examples, which added perturbations η using the “sign of the elements of the gradient of [loss] with respect to the input,” which is defined as

$$\eta = \text{sign}(\nabla_x J(\theta, x, y)) \quad (1)$$

where x is the input to the model, y is the target of x , and $J(\theta, x, y)$ is the cost used to train the network.

Rozsa et al. [1] extended upon the FGS approach for generating adversarial examples to demonstrate two effective ways to produce more robust adversarial images using fast gradient value (FGV) and the hot/cold approach. The fast gradient value (FGV) approach uses “a scaled version of the raw gradient of loss” to create adversarial examples with distortions even less perceptible to humans. The direction of this type of perturbation η_{grad} is defined by

$$\eta_{\text{grad}} = \nabla_x J(\theta, x, y) \quad (2)$$

where θ is the parameters of the model, x is the input of the network, y is the label of x and $J(\theta, x, y)$ is the cost used to train the network. The hot/cold approach defines a hot class as the target classification class and a cold class as the original classification class. This method then uses these defined classes to create features that cause classification to move towards the hot or target class. In addition to

the different approaches to generating adversarial examples, Rozsa et al. also defined a metric for quantifying adversarial examples by measuring both the element-wise difference and probability that the image could be a different perspective of the original input called a Perceptual Adversarial Similarity Score (PASS). This score is a number between 0 and 1, where 1 denotes an adversarial example with no visible difference from the original image.

Rozsa et al. [1] also explored the effectiveness of fine tuning a DNN with adversarial examples and showed that such networks were able to correctly classify 86% of previously adversarial examples.

Rozsa et al. [2] also contributed to the known information about adversarial images by defining adversarial examples that exist in nature as “an image that is misclassified, but that will be correctly classified when an imperceptible modification is applied.” Natural adversarial images demonstrate an additional aspect of the security threat of adversarial examples that needs to be considered.

In response to the growing interest and research in adversarial examples, Papernot et al. [3] asserted that by using deep learning algorithms, system designers made security assumptions about DNNs, specifically in reference to adversarial samples. Papernot et al. [3] addressed the problem by demonstrating the use of distillation as an alternative form of training a DNN to increase the percentage of correctly classified adversarial examples when presented to the DNN as inputs using the CIFAR-10 [11] and MNIST [9] data sets. This approach increased correct classification of adversarial examples with the increase of distillation temperature, reaching a maximum of 99.55% correct classification on MNIST adversarial examples and 94.89% on CIFAR-10, both with a distillation temperature of 100. Each set of adversarial examples were crafted using the approach described in [14].

In addition to putting forward a new way to conquer the effect of adversarial examples, Papernot et al. [12] also demonstrated a method for attacking a DNN with adversarial examples without prior knowledge of the architecture of the network itself, and only having access to the targeted network’s output and some knowledge of the type of input. To accomplish this, Papernot et al. trained a substitute DNN on possible inputs for the targeted DNN. After the network was trained, adversarial examples were crafted with Goodfellow et al.’s FGS method [5]. These examples were generated with different values for ϵ as defined in Equation 1. Examples of the FGS adversarial examples generated with the varied values of ϵ can be seen in Figure 2. The examples generated for higher values of ϵ do not fit the portion of the definition of an adversarial example that the perturbations to the image image is imperceptible to a human.

A different technique for increasing a DNN’s ability to handle and correctly classify adversarial examples was put forward by Luo et al. [15]. This technique uses a “transformation of the image that selects a region in which the convolutional neural network (CNN) is applied, denoted a foveation, discarding the information from the other regions” as the input to the CNN. This technique enabled the CNN to correctly classify over 70% of adversarial examples.

The state-of-the-art [6], already takes crops of the original input and the average or median of the crops are then used as the input to the DNN, mimicking perturbations from natural input. A recent GoogLeNet DNN used 144 crops [18]. Crops like the ones used in the state-of-the-art predate the discovery of adversarial examples and are used to improve accuracy of DNNs. The improvement of accuracy in the DNN also applies to the increase of accuracy in classifying adversarial images. By taking crops of images, the accuracy of the DNN when classifying adversarial examples is greatly increased. The networks tested by Papernot et al. [3] did not use this state-of-the-art. This research aims to disprove Papernot et al.’s assertion that adversarial examples are a major security threat to DNNs.

III. METHOD

The proposed security threat provided that in a critical situation, the incorrect classification of an adversarial example made by the DNN could cause actions, which were taken based on that classification, with immense repercussions.

The image acquisition process as described above, which is necessary in all real world applications of classification by a DNN, is always going to capture an imperfect input.

To assess the extent of the potential security threat that adversarial examples cause for DNNs after acquisition, it is proposed that a slight transformation, such as a blur or a shift that would occur in normal image acquisition, be applied to images before they are classified by the DNN. In order to assess the extent to which adversarial images could survive the natural image acquisition process, a trained deep neural network, a dataset including adversarial examples, and transformations are used.

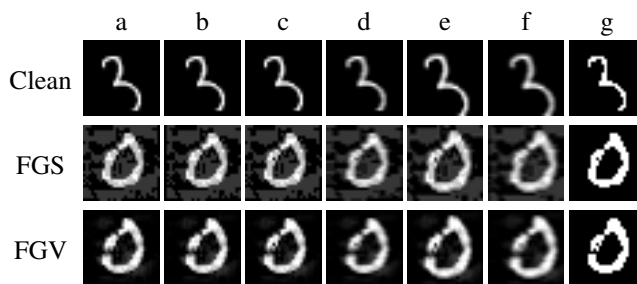


Fig. 3: **Transformations of Clean, FGS, and FGV Images**
The rows show transformations of a clean image, FGS adversarial and FGV adversarial. Column (a) is the original image. Column (b) has had one column translated to black. Column (c) has a small amount of noise to it. Column (d) has the blur kernel of (2,1) applied to it. Column (e) has been cropped 1 pixel by 1 pixel and then resized to 28 pixels by 28 pixels. Column (f) is a combination of all previously mentioned transformations. Column (g) is the result of binarization with Oshu thresholding.

A. The Deep Neural Network

A LeNet [13] deep neural network, trained by the authors of [1], is used for completing classification experiments on the chosen dataset for this research. As listed in Table 1, this network classifies images in a normal test set (with no adversarial examples) with an accuracy of 98.96%.

B. The Dataset

The DNN is trained on the MNIST dataset of handwritten digits [9], and will be tested with the MNIST test set. This dataset provides a basis for a possible security weakness of the DNN to adversarial examples. If adversarial examples cause an incorrect classification of a handwritten selection, such as an amount on a check, it could cause the amount to be misinterpreted and cause an incorrect amount to be withdrawn.

In addition to the MNIST test set [9], adversarial examples generated using the techniques in [1] and [5] are also tested to initially demonstrate the effectiveness of the adversarial example, and then to assess the effectiveness of the transformations, which mimic the acquisition process, at negating the effect of the perturbations that create an adversarial example.

In initial acquisition, MNIST images were subject to a pipeline of downsampling to 20x20 pixels, binarization, and subsequent upsampling to 28x28 pixels. The adversarial examples used in the following experiments were generated by the authors of [1] for the network described previously. When generating FGS adversarial examples, the authors of [1] stepped ϵ , as defined in Equation 1, until the image was made adversarial, so the entire data set is originally adversarial for the network described above.

C. Transformations

The aim of this research, is to assess the extent to which adversarial examples can be handled and classified correctly simply by the imperfection of the natural image acquisition

process such as slight transformations and the slight perturbations added to the images input for classification and understanding the impact each of these transformations has on classification. In order to mimic the image acquisition process as accurately as possible, an image was printed out, scanned back in, and analyzed for the types of transformation needed to closely replicate the effect of the acquisition. The types of transformations and perturbations that have been used to complete an experiment thus far and their justifications are listed below.

1) *Translation*: The addition of a translation to the images processed by the DNN replicates an alignment issue that could occur in the normal image acquisition process. This was implemented by shifting the image to the right by one pixel and filling the pixels in the empty column with values of 0, which in RGB values is black.

2) *Noise*: In the image acquisition process, it is normal to see additive noise on an image, such as black dots seen when an image is scanned in. To replicate the additive noise on an image, a small amount of computer generated noise is added to an input image before allowing the image to be processed and classified by the DNN. The noise mask was generated with a standard deviation of 0.25, and a mean of 0. The noise mask was then added to a copy of the original image.

3) *Blurring*: When acquiring an image in a real world environment, such as with a camera or scanner, it is virtually impossible to capture an image without any blurring. To replicate this, the amount of blurring seen in the image analyzed for transformations was estimated. This led to the application of a blur kernel of (2,1) was applied to the input images, as there was approximately one pixel of blur in the x direction and one-half pixel of blur in the y direction on the acquired image, with the asymmetric probably due to the scanner having a moving linear sensor.

4) *Cropping & Resizing*: This transformation mimics the event where when an image is acquired, it is smaller than the original image. In order to mimic this, input images were cropped, the cropped image was saved, and then the cropped image was resized using a cubic interpolation function back to the image size expected by the DNN of 28 pixels by 28 pixels.

5) *Combination*: The above transformations each demonstrate pieces of the whole image acquisition process. In order to fully synthetically capture this process, the described transformations must all be applied to the input images. Transformations were applied in the following order: translation, noise, blur, crop and resize. This order was chosen to mimic the order in which the transformations occur in the natural image acquisition process. After the transformations were applied, the transformed image was input to the DNN for classification.

D. Fine Tuning

The described experiments were run both on the raw LeNet [13] network, and on a fine tuned network. The network was fine tuned because when it was trained, the network

learned from clean images without transformations. When inputting transformed images for classification, the network was not robust enough to correctly classify transformed inputs, even if the inputs were not adversarial examples, with the same amount of accuracy as with clean images.

In order to fine tune the network, a set of 100,000 images was taken for training and 20,000 images were used for validation. The fine tuning sets contained a total of 60,000 clean images and 60,000 transformed images, where the transformed images were the MNIST training set images with the combination of all transformations was applied. The fine tuned network had an accuracy of 99.35% on the chosen validation set, and 99.09% on the MNIST testing set.

E. Fusion of Crops

In order to more accurately mimic the state of the art deep neural networks [6], a series of crops was implemented. As was previously mentioned, crops are used to increase the accuracy of DNNs. With only a 28x28 image for MNIST, the number and size of crops is more limited, so we used only 5 crops: a center crop of 26x26 pixels rescaled to 28x28 and 4 corner crops of size 27x27 rescaled to 28x28. Each implementation of resizing the image used a cubic interpolation function. Each crop was used as an input for classification by the DNN which in turn returned the vector of a score per digit label. The score vectors returned for all crops were added and the maximum value was used as the predicted label.

F. Binarization

As is common in hand-written text recognition [19], before applying the recognition engine the image is subject to preprocessing including binarization and noise removal. As mentioned above, the MNIST dataset [9] was subject to such preprocessing before being compiled into the dataset used in training and experimentation. The exact preprocessing of the MNIST images cannot be exactly replicated, due to an unclear description including lack of details on how down-sampling, binarization and subsequent up-sampling were performed. Without details of the rescaling steps, we approximate what we consider the most important step, binarization, using an OpenCVs [20] version of OTSU thresholding [21]. Binarization is a critical step and takes into account the fact that machines are trained on basically binary data. When it is forced to deal with data which is not binary, the machine is more easily confused. As we shall see, this single assumption may account for almost all the effectiveness of adversarial, and proper preprocessing renders them neutralized. Pure binarization may not be effective on the type of noise in [3], but the despeckeling/noise removal that is commonly used for document processing [19][22], would likely remove most of that noise as well.

IV. EXPERIMENTS & RESULTS

A. Procedure

The experimentation procedure involved three datasets: the MNIST test set [9], a set of 10,000 randomly chosen FGS [5]

Accuracy on MNIST Test Set		
Transformation	Raw Network	Fine Tuned Network
None	98.96%	99.09%
Translation of one column	94.95%	99.17%
Noise	98.95%	99.09%
Blur	98.70%	99.14%
Crop and Resize (1px x 1px)	98.35%	99.14%
Combination	97.66%	98.88%
5 crops (on non-transformed)	98.67%	99.12%
Binarize (on non-transformed)	98.76%	99.04%

TABLE I: This table reports the results of the experiments done thus far on the MNIST test set. Fine Tuned is the accuracy of the test set after the DNN was fine tuned on transformed images. Accuracy is based upon number of images classified correctly by the DNN.

Accuracy on 10,000 FGS Adversarials		
Transformation	Raw Network	Fine Tuned Network
None	0.00%	56.93%
Translation of one column	65.29%	68.93%
Noise	28.41%	59.84%
Blur	58.60%	59.83%
Crop and Resize (1px x 1px)	78.28%	80.01%
Combination	79.68%	83.98%
5 crops (on non-transformed)	90.94%	81.66%
Binarize (on non-transformed)	99.24%	99.21%

TABLE II: This table reports the results of the experiments done thus far on a set of FGS [5] adversarial examples. Fine Tuned is the accuracy of the test set after the DNN was fine tuned on transformed images. Accuracy is based upon number of images classified correctly by the DNN.

adversarial examples, and a set of 10,000 randomly chosen FGV [1] adversarial examples. All experiments were run on all three datasets in order to generate a basis of comparison for results. After running a baseline with no transformations, experiments consisted of applying a transformation or combination of transformations to input images and then passing the transformed inputs to the DNN for classification. The results of the classification were measured by the percentage of images correctly classified by the DNN.

B. Transformation Results

The specific results of the experiments completed are detailed in Tables 1, 2, and 3. These results demonstrate that the image acquisition process, allows the DNN to correctly classify a large portion of what used to be adversarial examples.

The transformation which is the most effective at allowing the DNN to compute a correct classification, is the cropping and subsequent resizing of an input image, which demonstrated 78.28% and 76.16% accuracy on the FGS [5] and FGV [1] datasets respectively for the raw network. This transformation also only slightly altered the accuracy of the raw DNN on the MNIST test set [9] (from 98.96% to 98.35%). Cropping and resizing also led to the highest accuracy on the fine tuned network, demonstrating 80.01% and 78.70% accuracy on the FGS [5] and FGV [1] datasets respectively and 99.14% on the MNIST test set.

Accuracy on 10,000 FGV Adversarials		
Transformation	Raw Network	Fine Tuned Network
None	0.03%	62.14%
Translation of one column	68.26%	73.15%
Noise	57.84%	64.59%
Blur	64.77%	65.54%
Crop and Resize (1px x 1px)	76.16%	78.70%
Combination	71.29%	75.95%
5 crops (on non-transformed)	81.66%	76.69%
Binarize (on non-transformed)	99.24%	98.88%

TABLE III: This table reports the results of the experiments done thus far on a set of FGV [1] adversarial examples. Fine Tuned is the accuracy of the test set after the DNN was fine tuned on transformed images. Accuracy is based upon number of images classified correctly by the DNN.

After fine tuning the network, the accuracy of classification of adversarial examples increased 56.93% and 62.09% on the FGS [5] and FGV [1] datasets respectively without applying transformations.

The results of this portion of the research demonstrates that in the majority of cases, the effect of perturbations added to make FGV adversarial examples are more easily negated than the perturbations added to make FGS adversarial examples. Intuitively, this is the case because the perturbations in FGS adversarial examples are bigger and more noticeable, and are therefore more likely to survive the transformations demonstrated in the natural image acquisition process.

It should be noted, that when transformations are applied, the performance of the deep neural network decreases on the MNIST test set. This decrease is the result of the added transformations essentially creating natural adversarial examples, as are defined in [2]. These natural adversarial examples introduce a different problem to the DNN, because when put in the same situation where an incorrect classification causes an incorrect action, the natural adversarial examples would also cause an incorrect action. Although these actions would not be chosen by an adversary, there would still be actions with consequences.

Although testing of the other transformations and combination of the transformations have not produced results where the transformation is completely negating the effect of the adversarial examples, all of the transformations have improved upon the accuracy rate of the DNN on adversarial examples without transformations.

C. Fusion of Crops Results

Doing experiments with the application and fusion of 5 and 10 crops produced the highest number of correctly classified adversarial images. Applying the crops mimics the methodology of the state-of-the-art [6] which uses crops to increase the accuracy of the DNN.

D. Binarization Results

Binarization produced the best results out of any of the transformations, achieving close to the performance of the deep neural network on the MNIST test set without any transformations. In the binarization process, more FGS adversar-

ial examples are correctly classified than FGV adversarial examples and thus are not surviving the image acquisition process. This is due to the fact that FGS adversarial examples depend on bigger and brighter collections of noise to render the image adversarial in comparison to FGV adversarial examples.

E. Adversarial Examples on ImageNet

After seeing the modest success of the synthetic image acquisition process on handling adversarial examples, an initial experiment was run on a GoogLeNet deep neural network [18] on a subset of the ImageNet dataset [17], with 15,000 FGS adversarial examples, all of which were provided by the authors of [1]. The experiment consisted of applying the combination of transformations, as is described above. The results of this experiments demonstrated that 63% of adversarial examples were classified correctly for top-1 accuracy and 89.95% of adversarial examples were classified correctly for top-5 accuracy. The fact that the application of transformations are producing similar results for the MNIST dataset and a portion of the ImageNet dataset demonstrates that foveation as described in [15] could be applied to any place in the image to negate the effect of adversarial examples.

V. CONCLUSION

The final goals of this project include assessing the extent to which adversarial examples are a security threat and demonstrating the effectiveness of simple solutions, such as slight transformations to the inputs, at mitigating that threat. This research has demonstrated that slight transformations do render the majority of input FGS and FGV adversarial examples as nonadversarial. The best results of this research, achieved through binarization of the inputs to the DNN, demonstrated performance near the performance of the deep neural network on clean images. This demonstrates that for the MNIST data set [9], the potential security threat is negligible, as the adversarial examples can be almost completely mitigated through binarization, which is part of the acquisition process of the original images.

Outside of the classification of handwritten digits, when considering an autonomous car, the camera capturing input for the DNN has the opportunity to capture a traffic sign hundreds of times, each at a slightly different angle, rotation, alignment and blur. This makes the chances of an adversary producing an adversarial example that would survive the image acquisition process significantly smaller than is suggested in research in related work. If, independently, each frame correctly classifies 90% of adversarial examples then to get a majority wrong, say 15 frames in 30 frames (1 second) would only have a $\binom{30}{15}(0.1)^{15} \approx 1.55 \times 10^{-6}$, i.e. about 1 in a million chance of causing an error.

However, further research should be focused on the effect of the natural image acquisition process on adversarial examples on a dataset such as ImageNet [17] in order to formalize and assess the extent of the possible security threat to deep neural networks in real world applications.

REFERENCES

- [1] A. Rozsa, E. M. Rudd, and T. E. Boulton, Adversarial Diversity and Hard Positive Generation. In CVPR Deep Vision Workshop 2016. arXiv:1605.01775.
- [2] A. Rozsa, M. Gunther, E. M. Rudd, and T. E. Boulton. Are Facial Attributes Adversarially Robust?. In IEEE International Conference on Pattern Recognition (ICPR) 2016. arXiv:1605.05411.
- [3] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks.
- [4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, Intriguing properties of neural networks, In ICLR, 2014. arXiv:1312.6199
- [5] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples, In *International Conference on Learning Representation (ICLR)*, 2015 arXiv:1412.6572.
- [6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, Caffe: Convolutional Architecture for Fast Feature Embedding, In *Proceedings of the ACM International Conference on Multimedia*, pp. 675-678. ACM, 2014.
- [7] D. Ciregan, U. Meier, and J. Schmidhuber, Multi-column deep neural networks for image classification, in 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 3642-3649.
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097-1105.
- [9] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits, 1998.
- [10] A. Nguyen, J. Yosinski, and J. Clune, Deep neural networks are easily fooled: High confidence predictions for unrecognizable images, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 4274-4279.
- [11] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.
- [12] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami, Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples, ArXiv e-prints, vol. 1602, p. arXiv:1602.02697, Feb. 2016.
- [13] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, et al., "Learning algorithms for classification: A comparison on handwritten digit recognition." in *Neural Networks: the statistical mechanics perspective*, 261-276, 1995.
- [14] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, The Limitations of Deep Learning in Adversarial Settings, in 2016 IEEE European Symposium on Security and Privacy (EuroSP), 2016, pp. 372-387.
- [15] Y. Luo, X. Boix, G. Roig, T. Poggio, and Q. Zhao, Foveation-based Mechanisms Alleviate Adversarial Examples, arXiv:1511.06292 [cs], Nov. 2015.
- [16] S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet, Adversarial Manipulation of Deep Representations, In *International Conference on Learning Representation (ICLR)*, 2016. arXiv:1511.05122 [cs], Nov. 2015.
- [17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, *Int J Comput Vis*, vol. 115, no. 3, pp. 211-252, Apr. 2015.
- [18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Going Deeper with Convolutions, In *International Conference on Learning Representation (ICLR)*, 2014. arXiv:1409.4842 [cs], Sep. 2014.
- [19] O. Nina, B. Morse, and W. Barrett, "A recursive Otsu thresholding method for scanned document binarization", in *IEEE Workshop on Applications of Computer Vision (WACV)*, 2011, pp. 307-314
- [20] G. Bradski, and K. Adrian. "Learning OpenCV: Computer vision with the OpenCV library". O'Reilly Media, Inc., 2008.
- [21] N. Otsu. "A threshold selection method from gray-level histograms." *Automatica* 11.285-296 (1975): 23-27.
- [22] S. Lu and C. L. Tan, "Thresholding of badly illuminated document images through photometric correction," In *ACM Symposium on Document Engineering*, 2007, pp. 38.

Implementing Machine Learning Opportunities in Elementary School Settings

Simon Ellis, Andrea McGeorge and Celeste Puccio¹

University of Colorado, Colorado Springs

Abstract:

Machine learning has become widespread in recent years. It is more often discussed on university campuses in computer science and engineering classes rather than in elementary schools. In studying educational applications of machine learning, the research is limited in scope for the elementary level. We have explored and researched this topic in order to develop and create both staff presentations and differentiated learning engagements for (k-5) students that begin to teach foundational concepts of machine learning and create greater excitement for computer science amongst elementary-aged children. It has been our intent, with this study, to develop learning engagements using some of the basics of the machine learning process in building knowledge and skills in young students.

Index Terms:

machine learning algorithms: a method of organizing data, giving computers the ability to learn without being explicitly programmed.
data mining: computational process of discovering patterns in large data sets.
constructivist learning: students construct understanding and knowledge through experiencing & reflecting on those experiences.

I. Introduction

“Machine learning is a computer’s ability to learn from data, and one of the most useful tools we have to develop intelligent systems and applications ... Data mining extracts information and finds patterns that can then be processed and communicated.” –Geoffrey Gordon, Carnegie Mellon University.

Machine learning is used widely today for all kinds of tasks, from a web search, to voice commands, to robotics. It’s hard to find an avenue that cannot benefit from machine learning in one way or another. Machine learning’s intuitive, versatile, and focused approach to finding patterns in available data, and directing responsive reactions, makes it an asset for, as of yet, an unlimited number of applications. In today’s world, it is more accessible than ever before, thanks to the variety of technology capabilities. Over the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search tools, and unique identification applications. To consider the information sorting ability it has provided for us in our everyday lives includes the benefit of email spam filtering, data delivery, preferential advertisements, and pattern/image recognition [Witten, 1999].

Data mining is the process of discovering patterns in data ... patterns must be meaningful and lead to an advantage when the data is in substantial quantities. “We are overwhelmed with data, every choice

we make is recorded (and these are just personal choices ... but also in commerce and industry choices). People frequently use data mining to gain knowledge, and not just predictions. (Witten et al. 2005)

The widespread application of the capabilities of machine learning in the educational setting have not yet been fully explored, nor have the responsibilities of these digital explorations been explained in their learning. Additionally, students today, no matter how unaware they may be, as to how a search engine is driven by machine learning, need to consider the underlying ramifications of soliciting or utilizing data derived through computer applications, both academic and those of social media.

The “Disinhibition Effect” gives students and adults alike a false sense of security, making them both comfortable and disarmed when sharing data online. (*BBC Radio4 –Technology July, 2016*) Our children are riding an implacable wave of technology! Whenever they go online, search the internet, send an email, keystroke on their computer, or make a phone call, their data can be organized and extrapolated to give important information to anyone who needs it, and writes code. This gives rise for the need of students to be aware of the consequences of their choices, and the reasons for cyber-security in this technological world of wonder.

II. Motivation

Introducing computer science at the foundational elementary level will bring about computer science awareness, with the long-term aim to increase the number of American citizens and permanent resident undergraduates who are attracted to careers in research & advanced studies in Computer Science. (Computer and Information Science and Engineering RET Supplements REU Sites)

Machine learning has been growing in importance in our lives and more importantly in our children’s lives, as the profusion of devices increases each school year. It is a pressing issue for our students to both learn and be aware of both the benefits of data mining and the downsides of what that connection to the virtual world can mean. Children need to be aware that when they type data into the computer, that data can be analyzed by data mining, revealing more information than intended or that they are aware of. Each lesson will include the concept of awareness of the negative effects of data mining.

By analyzing Google Trends, it can be determined how often a particular search term(s) has been used. Internet searches for the terms “data mining” & “machine learning” reveal a trend. Data mining had been searched online more consistently than machine learning over the past fourteen years. However, over the last twelve months, there has been an increase in machine learning search requests, and they have surpassed requests for data mining. This concludes that people’s interest in machine learning is growing as concerns for data mining are lessening. The graph below, figure 1, shows search requests for these phrases (‘data mining’ -top line, ‘machine learning’ -bottom line), since Jan 2004 until June 2016.

¹ Ellis, McGeorge and Puccio teach at Academy International Elementary School, in Academy School District 20, Colorado Springs, Colorado, USA. This work was supported by the 2016 Research Experience for Undergraduates (REU) program at the University of Colorado Colorado Springs (NSF Award No. 1359275).

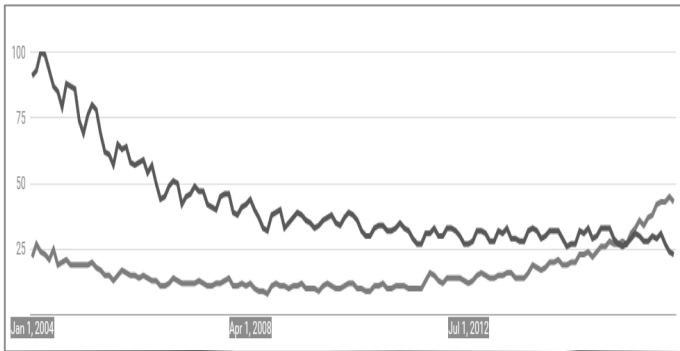


Figure 1. Search Request “Machine Learning & Data Mining

The Google Trends graph referenced that comparing the popularity of the two search terms “machine learning & data mining”, also has the ability to display which countries in the world those searches were made from. Since 2004 until the present, the top six countries in order of search popularity have been for:

“Data Mining”: Ethiopia, India, Nepal, Sri Lanka, Kenya, Singapore, and Hong Kong. See figure 2.

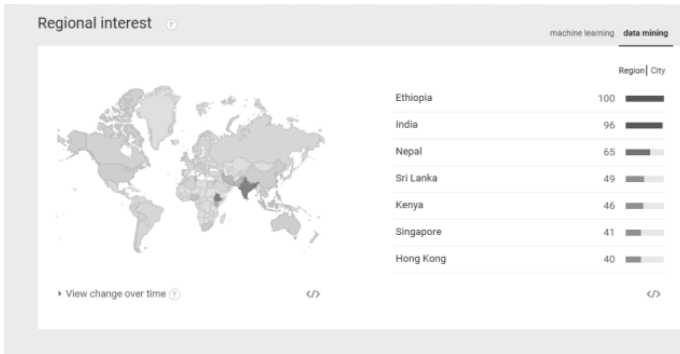


Figure 2. Search Request for “Data Mining” by Country

“Machine Learning”: South Korea, India, Singapore, Hong Kong, Israel, Bangladesh, and Pakistan. See figure 3.



Figure 3. Search Request for “Machine Learning” by Country

Educational delivery of data mining, as demonstrated through computer algorithms, is an area that has been limited to individualized linear completion of tasks and assessments, within certain curriculum content [Rose& Rush, 2009]. For students to embrace the delivery of lesson instruction about computer science, it needs to be inviting, provocative, and self-motivating.

It is these gaps in the development of meaningful, interactive, practical applications that need to be investigated and developed to assure that students are participating in 21st century learning, and ready to embrace their futures. As written previously in this paper, educators need to remind students of their vulnerability in searching for, and in sharing data about themselves to the world, through the internet, regardless of their intent.

It is our intent, with this project, to develop a framework for learning in parallel with current standards of what modern day technology can do for students of all ages, and to scaffold instructional modules to allow for developing the mindset in ways that elementary educators today can only imagine. It is only now that new classroom innovations are being developed to utilize the science in those school settings.

The Computer Science Teacher Association Standards (CSTA) last revised in 2011 by The CSTA Standards Task Force laid out the following:

- a) all students should have access to rigorous and culturally meaningful computer science and be held to high expectations for interacting with the curriculum.
- b) diverse experiences, beliefs, and ways of knowing computer science should be acknowledged, incorporated, and celebrated in the classroom.
- c) the integration of different interpretations, strategies, and solutions that are computationally sound enhance classroom discussions and deepen understandings.
- d) the resources needed for teaching and learning computer science should be equitably allocated across groups of students, classrooms, and schools.
- e) classroom learning communities should foster an environment in which all students are listened to, respected, and viewed as valuable contributors to the learning process.
- f) ongoing teacher reflection about belief systems, assumptions, and biases support the development of equitable teaching practices.

The CSTA proposes a three-layer model for K–12 computer science that addresses the needs of the present and future by building on the lessons of the past. It focuses on fundamental concepts with the following five complementary and essential strands:

1. Computational Thinking
2. Collaboration
3. Computing Practice and Programming
4. Computer and Communications Devices
5. Community, Global and Ethical Impact

In figure 4 below, the process is illustrated as a cyclical experience of learning as do the learning engagements in the module presented in the appendix. These concepts drove our inquiry and development of the ideas featured in these lessons. Ultimately, these concepts will result in lifelong skills that will be applied in the computer science world of the students involved, who potentially are our future computer science leaders.



Figure 4. CSTA :Five Fundamental Concepts of Cyclical Learning

The CSTA divides up the k-12 continuum into three levels /ages: Level 1 (recommended for grades K–6) “Computer Science and Me”. Level 2 (recommended for grades 7–9) “Computer Science and the Community”.

Level 3 (recommended for grades 10-14) is split into three areas of importance: ‘Computer Science in the Modern World’, ‘Computer Science Concepts and Practices’, and finally ‘Topics in Computer Science’.

The aim of the CSTA is that elementary school students are introduced to foundational concepts in computer science by integrating basic skills in technology with simple ideas about computational thinking. This parallels the educational spectrum of learning which students already experience throughout the curricular areas of literacy, mathematics, social studies and science. The learning experiences created from these standards will be inspiring and engaging, in so doing it will help students see computer science as an important part of their future world. The module of experiences is designed with a focus on active learning, creativity, and exploration. elementary, middle and high school. They have been used as a guide

The levels used by the CSTA gave us the scope and sequence in designing the module. (See appendix) These levels will also be used to help guide after school programs that explore computer science and machine coding. The focus of this paper is on the “Level 1” - elementary students K-6. However it should be pointed out that the school where this research will be primarily implemented is a K-5 institution.

Although these standards are thorough and were reviewed in 2011, they have still not been adopted by all states. This creates an obstacle of bringing computer science programs and standards to the k-12 level. Although the premise of this paper is to educate at a foundational level, for computer science at an elementary level, change needs to happen at the state level, so that standards can be integrated into the curriculum. However, events such as the upcoming Hour of Code in December, give hope for schools to offer meaningful computer science opportunities to the k-12 audience.

Community, Global, and Ethical Impacts

The ethical use of computers and networks is a fundamental aspect of computer science at all levels and should be seen as an essential element of both learning and practice. As soon as students begin using the internet, they should learn the norms for its ethical use.

Academy School District 20, and CSTA talk extensively about the principles of personal privacy, network security, software licenses, and copyrights that must be taught at an appropriate level in order to prepare students to become responsible citizens in the modern world.

CSTA talks about students being able to make informed and ethical choices among various types of software such as proprietary and open source and understand the importance of adhering to the licensing or user agreements. In an elementary school, the level and exposure of computer usage is closely monitored, therefore this would become more of a concern at middle school. Students should also be able to evaluate the reliability and accuracy of information they receive from the internet. Throughout their educational experience students are presented with these challenges, and taught what best practice is, not just in technology. These standards introduce elementary school students to foundational concepts in computer science by integrating basic skills in technology with basic concepts about computational thinking.

The learning experiences in this module created from these standards are intended to be inspiring and engaging, helping students to become enthusiastic about computer science and see computing as an important part of their world. They are designed with a focus on active learning, creativity, exploration, inquiry and are typically embedded within other curricular areas such as social science, literacy, mathematics, and science.

III. Problem Definition

As a direct result of many states not adopting the CSTA standards, in conducting research of elementary educational models currently in use, there appears to be limited curriculum for computer science, specifically machine learning. Therefore, the focus of this study has been on building an understanding of the basic elements of computer concepts and applications for elementary age students in their daily lives.

Most recently, under new initiatives, funded through governmental agencies (federal, state and local) there has been an increase in both public and privately supported program development, with the intent of exposing students to, and providing avenues for, implementation of scientific and computer explorations, intentionally intended for younger populations. It is the hope, nationwide, that through these engineering based programs, students’ interest will be piqued, and potentially open doors for further study. Admittedly, within the past few years, commercial programs have been developed that utilize the concept of machine learning and programming to instruct and entertain populations of all ages. However, there is much room left to establish a dedicated curriculum to lay the foundation for, and leave room for, the creativity embedded in uses of technology for elementary students. Lastly, in reference to initiatives sponsored by governmental agencies, (STEM/STEAM/National Science Foundation) great gains have been made in bringing the basics of science into assorted classrooms around the nation, with the intent to reduce the mystery and apprehension of using components of computer engineering in commonplace applications, and for all populations.

As mentioned earlier in this paper, computer science, more specifically machine learning, is a subject area that generally is not introduced at the elementary or even the middle school level. As participants and educators in our rapidly developing technological world, we are in a position to expose our students to material to help them build opportunities to access the global community, and its vast opportunities to build knowledge. With machine learning as the basis for developing certain science applications and predictors of results

in project based learning, we are empowering them to learn of the potential that technology holds in driving the future.

The primary research focus of this study, then, has been to investigate applications of technology relevant to the world of an elementary age student, and to provoke their curiosity as to, not only how it works, but how they could apply these new machine capabilities into their world.

It is also the authors' objective to insure students understand safety in the access and implementation of shared knowledge, within this world wide web of data. As teaching models, it is imperative that students are reminded to use caution when using their devices at school, as well as at home. Students also need to learn about how their digital footprint affects their lives, and to become aware of the rights and responsibilities of digital citizens.

In "Future Innovations in Science and Technology" (2003), Joseph Coates comments on the dramatic effect information technology will have over the next 25 years. Included within this broad swathe of a category are technologies promoting intelligence in systems and devices such as robots and virtual reality machines. Coates predicts that virtual reality (and other technologies) will cause massive changes in how we educate our students. Here we are halfway through that time-span and we are researching how best to present and teach elementary students about machine learning and data mining. (p252, Appendix C: The Concept of Change, "Patterns of Change" –College of William and Mary)

IV. Proposed Solution

By exploring recent developments in information gathering, voice controlled devices, and computer generated responses based on the premise of artificial intelligence, we've developed practical instructional modules to introduce students to some of the potential embedded in technology today. These lessons begin with building student curiosity as to the technological components of the programs they are familiar with, allowing hands-on provocation opportunities as part of the learning, in consideration of adapting to suit a range of elementary ages.

We've created a module which will introduce students to computer science concepts, specifically machine learning using the pedagogy of constructivist learning. What is meant by constructivist learning or constructivism? The term refers to a philosophy of education that learners construct knowledge for themselves. Each learner individually (and socially) constructs meaning, as they learn. The core ideas expressed by it have been clearly enunciated by John Dewey among others. "Constructing meaning is learning; there is no other kind. The dramatic consequences of this view are twofold; 1) We have to focus on the learner in thinking about learning (not on the subject/lesson to be taught). 2) There is no knowledge independent of the meaning attributed to experience (constructed) by the learner, or community of learners." [Institute for Inquiry, Exploratorium, San Francisco]. Constructivism is inquiry-based. If the students are asking questions, they are constructing meaning. It is our hope to help the students construct meaning by first connecting with their own background experience, which will then be used as a springboard for new connections and differentiated learning opportunities.

To begin, we gathered and created resources that helped build skills and confidence and energize the classroom with learning-by-doing opportunities. One of these was coding. "You can learn skills at any age but why wait when we can teach everyone to code now!"

(Richard Branson Virgin Group) I. "Hour of Code", (5th-11th December, 2016). The 'Hour of Code' is nationwide initiative by Computer Science Education Week [csedweek.org] and Code.org [code.org] Its aim is to introduce millions of students to one hour of computer science and computer programming. This will allow students opportunities to code, including but not limited to: "Star Wars: Building a Galaxy with Code", "Minecraft Hour of Code", and "Code with Anna and Elsa".

Additional provocations and learning engagements will alert the natural curiosity of school age children. PLTW, Project Lead The Way, provides a transformative learning experience for K-12 students and teachers across the US. It catalyzes lifelong interest in STEM. Through "PLTW Launch" K-5 students & teachers learn and discover code together. [pltw.org] (UCCS is an affiliate institution PLTW)

Another practical exploration is the MaKey MaKey circuit boards. These are products from collaboration between SparkFun and Jay Silver/Eric Rosenbaum of the MIT Media Lab. They are "Invention Kits for Everyone." They enable the learner to play Mario on Play-Doh or Piano with Bananas. These circuit boards allow students to begin to understand the how and why of the connection of computer coding through making actual electrical connections which illustrates the way a computer behaves.

Another product in a similar vein is the Raspberry Pi. It is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation with the intent to promote the teaching of basic computer science in schools and developing countries. "Scratch" (is) included as standard with Raspberry Pi and it enables anyone to start programming within minutes, without any prior knowledge." (Phil King, The MagPi magazine). Python computer coding language is taught for students to program with.

Websites such as Tynker.com, also offer the chance for students to build and play their own game. "Tynker is a creative computing platform where millions of kids have learned to program and build games, Minecraft mods, apps and more. Tynker offers self-paced online courses for children to learn coding at home, as well as an engaging programming curriculum for schools." (Tynker.com)

Cleverbot and Chatbot apps learn from answers to a "conversation it has, using machine learning embedded in its code." These apps will be perfect inquiry for students:

- How do they work?
- How do they learn?
- How do they know what to ask?
- What is it going to ask next?

There are also Chatbots using avatars such as –"Eviebot, Boibot, Chimpbot & Willbot". These apps will be introduced as provocations, but will be revisited as assessments at the end of the module to evaluate if students can share how these robots relate to machine learning.

Another resource which yielded positive insights into machine learning was the viewing of TED Talks for machine learning. It revealed the work of Prof Vijay Kumar at the University of Pennsylvania, his talks on "The Future of Flying Robots", and "Robots that Fly...and...Cooperate" helped demonstrate the extent to which this area of study and development has now reached with Prof Kumar, where he is the Dean of the School of Engineering and Applied Science. Daniel Ueda at Penn Engineering and the GRASP

Lab (General Robotics, Automation, Sensing and Perception Lab), helped the research with links and resources that his lab has used for similar research projects. Students are always readily engaged in lessons that involve robots.

CSUnplugged.org is another resource that “introduces computer science basics in a format that’s fun & accessible to the youngest learners (grades K-5)” (code.org) “Children in their formative years (of elementary school), love learning through play.” [qubizm.co.uk]. The easy & fun activities in this book, designed for students of all ages, introduce you to some of the building blocks of how computers work—without using a computer. “This book can be effectively used in enrichment and extension programmes, or even in the regular classroom. You don’t have to be a computer expert to enjoy learning these principles with your students.” (CSUnplugged). The book contains a range of activities, with background information explained simply. We have adapted some concepts and activities from this resource to help build our module as well.

The module we created can be found in the appendix. In the first lesson, students are presented with practical applications of machine learning they may be familiar with, such as iPads, smartphones, robots, drones and AI apps. Many of us use computers every day, but how do they work? How do they think? How can people write software that is fast and easy to use? Do machines make mistakes? Computer science is a subject that explores these very questions. This is the provocation in the first lesson. After this provocation, we have the students try to find patterns in translating words and phrases as a means of introducing data mining.

To introduce decision trees, we’ve designed a lesson that introduces a guessing method that predicts what the user is going to type next, as in machine learning. The computer suggests what it thinks the students are likely to type next, and they just indicate what they want. This sort of system is also used to ‘type’ texts on some cell phones.

Another learning engagement that introduces concepts of networks, shows how computers use networks to sort. This is a kinesthetic activity that the children can walk through to understand better.

We introduce another activity or game, the orange game, to demonstrate that sometimes you have to streamline data to make it work well in a network. There are problems in many networks, especially when data mining. Computer scientists spend a lot of time figuring out how to solve these problems and how to use networks that make the problems easier to solve and patterns easier to detect. If you can figure out how to make problems easier, they are more efficient.

The activity “Conversations with Computers” aims to stimulate discussion on the question of whether computers can exhibit “intelligence,” or are ever likely to do so in the future, which is still a controversial topic that can lead to critical thinking and stimulating classroom discussions. Our hope is to convey to the students that machines make mistakes and we should always be critical thinkers.

The module culminates with a Vocabulary Jeopardy game, which is a review of the concepts and vocabulary taught in the module. This game will function as an assessment as well as an engaging culminating lesson.

V. Conclusion

At the conclusion of this research, developmentally appropriate, and somewhat guided instructional materials, have been tailored to curriculum appropriate for younger students. In learning the basis of

computer programming and indeed, how a computer can actually perform the functions that it is capable of, the content of our lessons introduce basic understandings, and allow students to build their own investigations of how this tool of today can extend the capabilities of research and science of tomorrow.

Through the elements of precise and sound instructional design and implementation, and in exploring the documented advantages of constructivist learning, students today even at a young age, are in a position to grasp to some degree, the processes in technology that support the world as they know it. It is only with creative and provocative delivery that this content will become part of their thinking and lay the groundwork for future science based developments and inspirations.

Presently, all curriculums are standards-based and when states adopt computer science standards, the students will benefit from our original objective of getting more people involved in this movement towards computer science in the school.

Upon conclusion of this module, the students will be aware of machine learning and data mining and be able to express the implications, both positive and negative of this relatively new area of study.

In addition, we aim to provide meaningful presentations for school staff about both machine learning and data mining, so that they too can introduce some of these lessons or any of the concepts to their students. The expectation is that these learnings will enhance the technological awareness of this primary/intermediate age-group, which in turn, will increase the number of American citizens and permanent resident undergraduates who are attracted to careers in research and advanced studies in computer science.

Acknowledgments: Thank You to the following people for their help, questions and assistance with this research project; Dr. Jugal Kalita, Dr. Terry Boulton, Tri Si Doan, Cora Coleman, Andrea Costas, David Foley, Abigail Graese, Nathan Harmon, Kamal Kamalaldin, Alanya Kennedy, Liam Schramm, Matthew Simpson, and Kyra Yee.

VI. References:

- [1] Witten, Ian H. (1999) Data Mining: Practical Machine Learning/Tools and Techniques. Pp.3-56.
- [2] Harrington, Peter. (2011) Machine Learning in Action. Pp.18-49.
- [3] Clifford, Miriam. “*20 Collaborative Learning Tips and Strategies for Teachers*”, July 1, 2016. Teach Thought, www.teachthought.com/pedagogy
- [4] “*Hands-On Activity- Cars: Engineering for Efficiency*”, May 2015. Teach Engineering: UC Boulder. www.teachengineering.org/lessons
- [5] Yonge, P.K. “*Blended Learning: Making it Work in the Classroom*”, September 11, 2014. www.edutopia.org.
- [6] Rose, Joel, and Rush, Christopher. Teach to One: School of One Program, September 15, 2009. www.edweek.org/digitaleducation

Author Index

Borhegyi, Zsolt.....	26
Boult, Terrence.....	44, 51, 57
Coleman, Cora.....	44
Costas, Andrea.....	51
Cserpa, Dorottya.....	26
Ellis, Simon.....	63
Erdi, Peter.....	26
Foley, David.....	57
Graese, Abigail.....	10
Harmon, Nathan.....	13
Kalita, Jugal.....	1, 6, 13, 16, 38
Kamalaldin, Kamal	26
Kennedy, Alayna.....	31
Lewis, Rory.....	26, 31
McGeorge, Andrea.....	63
Mello, Chad.....	26
Puccio, Celeste.....	63
Rozsa, Andras.....	57
Rudd, Ethan.....	44
Schramm, Liam.....	38
Simpson, Matthew.....	16
Yee, Kyra.....	6
Yi, Qing.....	13, 16
Zolta, Somogyvari.....	26

Keyword Index

Adversarial Examples.....	57
Algorithm Selection.....	16
Automatic Program Repair.....	26
Biomechanical Analysis.....	31
Brain State Clustering.....	26
Compiler Optimization.....	13
Compositional Distributional Semantics.....	6
Compound Noun Relation Classification.....	6
Computation Pattern Classification	13
Computational Linguistics.....	1
Constructivist Learning.....	63
Data Mining.....	63
Deep Neural Networks.....	57
Electromyogram.....	31
Elementary Schools.....	63
Epilepsy Localization.....	26
Fingerprints.....	51
Information Flow.....	26
Intrusion Detection.....	44
Machine Learning.....	63
Malware Recognition.....	44
Meta-algorithms.....	16
Myoelectric Control Schemes.....	31
Neural Networks.....	31
Neuroclustering.....	26
Nominal Compounds.....	6
Open Set.....	44
Partial Fingerprints.....	51
Prostheses.....	31
Resultant.....	16
Self-organizing Maps.....	31
Semantic Search.....	26
Shortest Tour.....	16
SIFT Features.....	51
Superfunctions.....	16
Spanish Word Embeddings.....	6
Sugihara Causality.....	26
Superfunctions.....	16
SVM.....	16, 44
Synonym Selection.....	1
Traveling Salesman Problem.....	16
Vector Semantics.....	1
WordNet.....	1
Wordsense Disambiguation.....	1