Semi-Supervised Labeling and Classification of Words by Semantic Subject

Uladzimir Kasacheuski
Department of Physics, School of Science
Department of Computer Science, School of Science
Indiana University-Purdue University Indianapolis. [*]

(Dated: May 2017)

The aggregation and organization of natural language text has garnered attention due to its utility in managing the vast amounts of data freely available today. Demonstrated in this work is an efficient method of utilizing the semantics embedded in word vectors to generate labeled data sets, words classified by subject, in a semi-supervised manner. For example, the words *quantum_mechanics*, *Einstein*, and *relativity* are all of the class *physics*. This labeled data is then employed in the training of supervised learning models such as neural networks and random forests, demonstrating its utility. The resultant data sets and models can then be applied as features for further classification, such as classifying text by subject or as a guide for generating hierarchical representations of the human language.

I. INTRODUCTION

The massive volume of digital natural language data freely available today is often an overwhelming amount to efficiently parse for a human, especially under time constraints. To this end, automated information aggregation and organization has been a topic of interest for many years [13, 26, 33, 36, 39, 44, 48].

The process of organizing information requires the segmentation of documents by subject. There is no utility in information of unrelated classes to be aggregated and summarized together. In other words, if a person is researching the various types of aquatic life present on earth it would be of little use to present a thorough summary on electric vehicles.

This work presents an efficient semi-supervised method of creating a labeled data set of words, classified by their semantic subjects, for any subject. Further, the utility of the data sets is demonstrated by employing them in supervised classification models. The ability to classify words based on semantic structure enables the usage of word subject in further classification, much like the results of named entity recognition and parts-of-speech tagging are utilized. Additionally, it demonstrates a feasible path to a hierarchical classification of the human language. For example, a *honda_civic* is of a class *car*, which is of a class *vehicle*, and so on.

II. RELATED WORK

Natural language processing (NLP) is a field with documented origins beginning in the late 1940's [25]. NLP focuses on tasks including machine translation, summarization, cross language information retrieval, speech recognition, dialog systems, information extraction, information retrieval, text categorization, and more [8, 25, 27]. The field has become prolific due to vast volumes of digital language data available through the Internet [8].

The information extraction application is particularly interesting due to its utility: it extracts information from natural language which can be further used in other NLP applications. Prominent examples include parts-of-speech tagging (POS)[41], named entity recognition (NER)[34], and word vector embedding [42].

The work presented in this paper leverages advancements presented by modern word embedding techniques, is relevant to tasks in NLP such as, notably, text categorization, and is conceptually similar in objective to NER.

A. Word Embeddings

A ubiquitous element in modern NLP work is the usage of word embeddings as features. Word embeddings are denser and richer feature representation of words generated by vector space models. More specifically, word embeddings are a sets of vectors that encode the semantic meaning of individual words to the end of enabling a machine to 'understand' the meaning behind words, similar to process humans employ.

The underlying concept of word embeddings is posited by the idea that "You shall know a word by the company it keeps..." [15]. The first vector space model documented is latent semantic analysis (LSA), which has been shown to preform on a human level at learning word semantics [24, 49]. Modern word embedding algorithms, such as the skip-gram model [30, 32] and later GloVe [38], continue to improve on the quality of these embeddings.

Part of the prevalence of this information extraction technique in modern NLP tasks can be explained by its wide relevance: not only can word vectors can be learned for domain specific data sets [14], trained and tested word embeddings are easily available available online [31]. The rest of the prevalence is likely due to the wide range of improvements word embeddings have brought to NLP tasks [11, 12, 37, 52].

[*] Code base for this project is publicly available at https://github.com/UladKasach/Word-Subject-Classification

B. Text Categorization

The ability to understand the semantic meaning of words, enabled by word embeddings, is a powerful tool for NLP researchers; however, it on its own will not solve most NLP problems. For example, in order to organize and aggregate volumes of content, a hierarchical understanding of word semantics is required. This goal of organizing text by categorical hierarchy is largely the domain of text categorization [43].

Most early methods of text categorization involve translating a body of text into a feature set which can then be passed on to supervised classification models. This feature selection process typically involves a vectorization of the text, a dimensionality reduction of the vectorized data, and data scaled according to heuristics [1, 19, 43]. Upon having a suitable feature set representing each document, standard supervised classifiers are employed [50].

Not surprisingly, word embeddings have, too, entered this domain, resulting in a different work flow than as in traditional text categorization methods. Generally they create a different feature space, utilizing word embeddings, and then train classifiers on the word embeddings feature space. These classifiers are often neural networks [20, 23].

It is relevant to note that word embeddings have not been the only information extraction technique utilized in text categorization. By increasing the quality of the feature space information extraction techniques can improve text categorization results; for example, POS[47] and NER[22] have been successfully applied in this context.

C. Semantic Subject Recognition

While NER and POS have progressed with the development of word embeddings, no information extraction algorithms have been raised which fully utilize the inherent semantic data encoded in word embeddings. To this end, it is likely that the extraction of semantic subject from a words vector embedding can be utilized as a relevant information extraction process.

This concept is then used in a semi-supervised approach to build a dataset of words labeled by semantic subject; an idea idea has been successfully employed before in NER domains [35]. These dataset can then be further used in generating models which classify word vectors by subject - generating additional features which can be utilized like POS and NER are in general NLP tasks.

NER is the conceptually closest task related to classification of words by semantic subject, being that its goal, too, is the classification of individual words based on semantics; however, due to the restriction of the domain to only those entities for which rigid designators exist [34], this work does not technically fall under that domain.

III. DATASET

This work required the use of well trained word vectors from which embedded semantic information could be extracted for semantic subject recognition. While initially the idea of training our own word vectors seemed feasible it was quickly[1] realized that aggregating the content required to train vectors off of, optimizing the tokenization of the content, and learning the vectors is a significant undertaking time wise both computationally and for the researcher. To this end pre-trained and tested word vectors were used from online sources.

In particular, this study used a word embeddings dataset[2] trained on a the Google News corpus, originally generated to demonstrate the success of two new word embedding algorithms, the continuous bag-of-words model and the continuous skip-gram model [30]. In their follow up work, this dataset was trained on an internal Google dataset of about 100 billion words with 3 million unique words and phrases[30, 31].

The performance of the dataset was measured by evaluating the accuracy of the embeddings on thousands of semantic and syntactic questions [30, 32]. The significant improvement in semantic accuracy of the skip-gram model in particular lead to further iteration and hyperparameter tuning of the Google News vectors and culminated in a dataset of word vectors which out performed prior work by a significant margin [31].

IV. ALGORITHMS

This work was based on many various algorithms employed for many various goals. The word embeddings used were generated by employing the continuous skip-gram model flavor of word2vec. Oversampling, under sampling, and SMOTE algorithms were employed to deal with the significant imbalance in data classes. Neural network (NN), random forest (RF), support vector machine (SVM), and k nearest neighbor (KNN) algorithms were implemented for classification. Precision-recall (PR) comparisons were conducted to analyze training performance.

A. The Skip-Gram Word2Vec Model

The continuous skip-gram model is one of the two models encompassed by the word2vec embedding architecture. This model is introduced in Mikolov et al. in

[1] The infrastructure built to convert text data (html, ebooks, txt files, etc) into tokens usable for training can be found in the project repository.

[2] The Google News word embeddings dataset can be found at https://code.google.com/archive/p/word2vec/ and can be referenced by the file name GoogleNews-vectorsnegative300.bin

2013 and was shown to capture semantic information better than all other prior models, including the second word2vec flavor, continuous bag-of-words (CBOW) [30].

The skip-gram vector space model is a predictive model for learning word embeddings, rather than a count-based model such as LSA and GloVe[2, 38]. This means that rather than counting the co-occurrence of words and mapping this statistic into word vectors, the skip-gram model learns word representations by attempting to predict the appearance of words given contexts.

Fundamentally, predictive language models are based on the premise that words are conditionally dependent on the words that came before them [5]. The assumption that words are more dependent on words closer to their usage can be used to significantly reduce the difficulty of this problem by only requiring the consideration of a specific context window, c, in prediction. Given that w_i^j is the sub sequence of words from index i to j, i.e. $w_i^j = (w_i, w_{i+1}, ...w_{j-1}, w_j)$, the premise and context assumption are described by the chain rule of probability in equation 1 [21].

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t|w_1^{t-1}) \approx \prod_{t=1}^{T} P(w_t|w_{t-c+1}^{t-1}) \quad (1)$$

Earlier predictive neural net models [3] suffered from calculating the joint conditional probabilities for n-gram word sequences [29], resulting in time complexities on the order of $O(HV)$ where H is the number of elements in the hidden layer used to compute conditional probability and V is the total vocabulary size[30]. To this end, the word2vec models avoids learning the joint conditional probabilities explicitly while still learning the word vector representations. This results in significantly reduced time complexities on the order of $O(cDlog_2(V))$, where D the dimensionality of the learned word vectors[30].

The goal of the skip-gram algorithm to learn word representations useful for predicting the words often found in the target word's context. For example, given the sentence "the cat meows" the skip-gram algorithm would train its vectors to predict that *cat* is found next to *the* and *meows*. [31].

Conceptually the goal of the word2vec models is very similar to that of auto-encoders [28]. The input layer and the output layer of the neural network are both V dimensional vectors. Each target word, context word tuple (e.g., *cat*, *meows*) is then trained on. The target word is passed as a one-hot encoded vector into the input layer and the goal of the modified neural network is to predict that the output is most likely the one-hot encoding of the context word.

The hidden layer of the neural network, size V by D, is then learned such that the output layer can best predict the context word. This hidden layer, like in an auto-encoder, is actually the part of the model in which we are interested. The resultant hidden layer encodes a dense representation of data which we use as our word embeddings.

B. Oversampling, Undersampling, SMOTE

Imbalanced datasets have long caused problems in classifiers for various reasons. A notable problem which arises is that given a positive class small enough, a classifier may learn that it can maximize its performance by simply assuming all data has a negative label. While one approach is to modify the cost functions of the classifiers to assign a higher penalty to classifying the positive class[46], this does not deal with the problem that there may be comparatively little data that the classifier will see of the positive class.

Fundamentally, oversampling and undersampling are very closely related and quite simple concepts. Oversampling is the process of duplicating positive class datapoints in the training data. Undersampling is the opposite; it is the process of removing negative class datapoints from the training data. While conceptually simple, these techniques have been proven to be effective the naive implementations have significant room to be improved on [7, 18].

One example of said improved implementations is the synthetic minority over-sampling technique (SMOTE)[6]. This technique generates synthetic minority class samples by selecting two nearby minority class datapoints and creating a new "synthetic" datapoint at a random position along a vector between the two points.

C. Neural Networks

The neural network architecture is loosely based off of how neurons work in the brain. Large quantities of neurons are interconnected in both structures. At every moment, each neuron receives a vector of input composed from the signals of other neurons, $\vec{x}$, and responds to this input by emitting its own signal, z. The ability of the neurons to strengthen or weaken the weight of the input from each specific neuron, designated by a vector $\vec{w}$, is what researchers believe enables neural networks to adapt and self program [9].

Mathematically the relationship between the input and output of a neuron is expressed in equation 2, originally named as a perception [16]. Typically, this output, a, is then passed into a non-linear activation function, such as the sigmoid function in equation 3. By enabling the neurons to generate non-linear outputs, the neurons have the capability of representing non-linear functions.

$$a = b_0 + \vec{w}\vec{x} = b_0 + \sum_{j=1} \vec{w}_j \vec{x}_j \quad (2)$$

$$g(a) = \frac{1}{1 + e^{-a}} \quad (3)$$

So far we have only described a single neuron. Full neural networks contain layers of multiple neurons which all receive the same input at the same time and are free to update their weights individually. This means that for each layer of neurons, size $n^{(i)}$, we have a matrix of weights, $\mathbf{w}^{(i)}$, size $n^{(i)}$ by $n^{(i-1)}$. The term $n^{(i-1)}$ corresponds to the number of neurons in the previous layer - or the number of inputs if it is the first neural layer. By combining multiple layers of neurons, a neural network is then capable of modeling non-linear functions such as the boolean *xor* function[45].

The training of neural networks resolves to the learning optimal values for the weights associated with the inputs for each neuron, the matrix $\mathbf{w}^{(i)}$. This is typically done utilizing the gradient descent approach with propagation throughout the layers. The idea is encompassed by equation 5 where α is the step size and $g(a) * \Delta$ represents gradient of error with respect to $\mathbf{w}$ [40].

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha * g(a) * \Delta \tag{4}$$

$$\frac{\partial Loss}{\partial \mathbf{w}} = g(a) * \Delta \tag{5}$$

In this work we utilize a two layer neural network with the number of neurons per layer left as a parameter. In addition to the number of neurons per layer, the class weight associated cost was a tunable hyperparameter in this implementation. The output layer of the network is two neurons which represent the probability that the class is positive and negative respectively. The network is built on the tensorflow platform.

D. Random Forest

Random forests are fundamentally ensembles of many different decision tree classifiers. The randomness comes from the selection of random feature subspaces to train each compositional decision tree upon[17] as well as the bootstrap aggregating method, otherwise known as bagging, which selects a random subset of the data upon which to train each decision tree[4]. The composition of each different decision tree enables the random forest model to generalize much better on complex datasets while decreasing the risk of over fitting.

Decision trees operate by determining axis-parallel hyperplanes which best separate the data by class. Each hyperplane partitions the data into two smaller sets of data[51]. The resultant data partitions are then recursively split with further axis-parallel hyperplanes until the purity, defined in equation 6 where $n_j = |\mathbf{D}_j|$ and n_{ji} is the number of points in $\mathbf{D}_j$ with class label c_i, falls below a chosen purity threshold. Split points are chosen based on how well they separate the data. A common split point evaluation measurement is information gain, defined in terms of entropy.

$$purity(\mathbf{D}_j) = \max_i \frac{n_{ji}}{n_j} \tag{6}$$

$$Entropy = H(\mathbf{D}) = -\sum_{i=1}^{k} P(c_i|\mathbf{D}) log_2 P(c_i|\mathbf{D}) \tag{7}$$

$$Gain = H(\mathbf{D}) - \frac{n_y}{n} H(\mathbf{D}_y) - \frac{n_n}{n} H(\mathbf{D}_n) \tag{8}$$

This work utilized the scikit learn implementation of the random forest classifier. Tunable hyperparameters used with this classifier included class weight.

E. Support Vector Machine

The basic objective of support vector machines is to find a hyperplane which maximizes the margin between classes[51]. An important aspect of support vector machines is their ability to implicitly map data into a feature space in which the data is linearly separable[10].

Formally, this objective function is defined by equation 9. $\vec{w}$ is a d dimensional weight vector and b is the bias. For context, the hyperplane is defined by all points, $\vec{x}$, where $h(\vec{x}) = \vec{w}^T \vec{x}_i + b) = 0$ and the distance from a point $\vec{x}$ to the hyperplane is $r = h(\vec{x})/|\vec{w}|$.

$$\begin{aligned} &\min_{\vec{w},b,\xi} \Big\{ \frac{|\vec{w}|^2}{2} + C \sum_{i=1}^{n} (\xi_i)^k \Big\} \\ s.t. \quad & y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i, \forall \vec{x}_i \in \mathbf{D} \\ and \quad & \xi_i \geq 0, \forall \vec{x}_i \in \mathbf{D} \end{aligned} \tag{9}$$

The ability of support vector machines to implicitly map data into a feature space where the data is linearly separable stems from the dual formulation of SVM objective function, seen in equation 10. In this formulation, the term $\vec{x}_i^T \vec{x}_j$ is simply computing the similarity between two vectors. By replacing this similarity function with any valid kernel function, $k(\vec{x}_i, \vec{x}_j)$, we are able to compute the similarity between two vectors in a different feature space without needing to explicitly know a transformation function which maps the data into that feature space.

$$\begin{aligned} \max_{\alpha_i} L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j \\ s.t. \quad 0 \leq \alpha_i \leq C, \forall i \in \mathbf{D} \\ and \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{10}$$

This work utilized the scikit learn implementation of the support vector machine classifier. Tunable hyperparameters used with this classifier included class weight, kernel, and kernel specific options such as the degree for a polynomial kernel.

F. K Nearest Neighbor

K nearest neighbor algorithms operate on finding the k data points which minimize the distance or maximize the similarity between the data point and a target data point. When used as a classifier the algorithm is a non parametric probabilistic classifier and it's objective function is defined by equation 11. In other words, the classifier predicts that the class of the target data point is the majority class among its k nearest neighbors [51].

$$\hat{y} = arg\max_{c_i}\{P(c_i|\vec{x})\} = arg\max_{c_i}\{K_i\} \tag{11}$$

This work utilized the scikit learn implementation of the k nearest neighbor classifier. The only tunable parameter utilized for this classifier was the value of k.

G. Precision, Recall, and F1 Score

Precision, recall, and F1 score are statistics commonly used when assessing the performance of a classifier. For a binary classifier, these statistics are defined in equations 12 - 14, where $n_{ij} = |\{\vec{x}_a \in \mathbf{D}|\hat{y}_a = c_i, y_a = c_j\}|$, $TP = n_{11}$, $FP = n_{12}$, $FN = n_{21}$, and $TN = n_{22}$.

$$precision = \frac{TP}{TP + FP} \tag{12}$$

$$recall = \frac{TP}{TP + FN} \tag{13}$$

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{14}$$

The F1 score is defined as the harmonic mean of the classifier. It attempts to balance the precision and recall values, where perfect recall and precision result in a maximum score of 1.

The trade off between precision and recall can be analyzed graphically by plotting precision on the y axis and recall on the x axis. This is referred to as a precision-recall curve.

V. CLASSIFICATION

Classification in this work is heavily interwoven with the iterative improvement of positive labels. On a high level, the process breaks down into a expectation and maximization (EM) paradigm. First, many different classifier-hyperparameter sets are trained on a set of labeled data and the predictions, i.e. expectations, of the classifier are recorded. Next, these results are then analyzed and the misclassifications are manually assessed to remove labeling errors, maximizing the accuracy of our labeled dataset. Like other EM processes, these two steps are repeated until the results converge.

A. Create Seed Dataset

In order to begin the process, we need to start with a labeled dataset, a dataset of 'seed' positive class labels. Rather than attempting to find all positive class words in the word vector dataset, a small set of words, ≈ 80 words, was used to efficiently generate a good approximation of the positive class.

For each of the ≈ 80 words manually defined to be of the positive class, the 250 nearest neighbors were found and recorded. These lists were then quickly checked to make sure that the words found were of the proper semantic subject. For example, the word *succulent* has two meanings: one in the context of *plants* and the other in the context of *food*. Words that generated lists of words relating to the wrong semantic subject were removed. The nearest neighbors for all of the 80 words that related to the proper subject were then compiled into one large list.

B. Iteratively Improve Labels

1. *Expectation*

Each expectation step in the iteration process consists of recording the classification results of many different combinations of classifier, hyperparameters, and sampling-parameters. First all desired sampling-parameters are used to generate test/train split labeled datasets which can be used for supervised classification. Then, the each classification parameter combination is used to train a classifier and evaluate its performance on the test data.

Generating each train, test data split requires several processing steps. First the dataset of word vectors must be labeled according to the dataset of positive class words for this iteration. After this is done, a random subset of words comprising a certain percentage of the labeled data is removed and recorded as our test data. Care is taken to ensure that the overall percentage of positive data in the labeled dataset and in the test dataset are the same. For example, if the positive class composes 5% of the labeled data, it should compose 5% of the test data. The rest of the data we designate as our training data. Before recording the training data, we first apply any sampling methods desired; i.e., oversampling, under sampling, or SMOTE.

Sampling-parameters specify both the sampling method to use as well as the sampling multiplier. For example, a sampling multiplier of 2 would result in the training data having twice as much positive class data.

A grid search[3] related array of parameter sets,

[3] Every possible enumeration of a discrete set of parameter values, including sampling-parameters, is generated, much like grid search enumerates hyperparameter sets to train with.

each of which specifies hyperparameters and sampling-parameters, was generated for each classifier based off of parameter combinations deemed of interest. Each parameter set was then used to train its respective classifier.

After training, each classifier is evaluated on their respective test data and the results are recorded. Recorded results include misclassification statistics and the specific words that the classifier misclassified.

2. *Maximization*

Each maximization step consists of summarizing the statistics of the classifiers and of assessing misclassifications in order to maximize the accuracy of the dataset of positive class labels.

The results of all classifiers are aggregated into one table of information listed in order of decreasing $F1$ score. This table of data was then used to plot precision-recall curves for each iteration as well as to compare classifiers and parameter sets.

The words misclassified by each classifier are aggregated in a list for both FP and FN in order of decreasing frequency of occurrence for each word. These lists are then used to manually double check the words most misclassified. Often, the words most commonly misclassified are misclassified because the positive label dataset is either missing a word that is actually of the positive class or includes a word that is actually of the negative class.

In this manner, the classifier detects words improperly labeled in an unsupervised manner and the labels are updated manually resulting in a semi-supervised approach to iteratively generating an accurate dataset of positive labels.

VI. RESULTS

This classification 'pipeline' was utilized with respect to the semantic subject of *plants*, in the context of *trees*, *shrubs*, *flowers*, *houseplants*, etc. The initial seed dataset consisted of $3.7k$ unique words, with $16k$ words repeated between the neighbors.

The sets of classification parameters for each classifier were kept constant throughout each iteration. The parameters defined resulted in 1809 different classifications in each iteration. Three iterations were conducted on the dataset.

Both the precision-recall curves, figures 1 and 2, and pre-maximization F1 scores, recorded in table I, show clearly that through each iteration the classifier performance increased in the pre-max evaluation. Table II shows the amount of change undergone by the true label set throughout each iteration. Table I lists the F1 score, after the maximization step, of the classifier with the highest F1 score before the maximization step. To achieve a better estimate of classifier performance, all misclassifications of the pre-maximization highest F1 score were checked for labeling errors. This is in addition to checking the top thousand most commonly misclassified words as done for the maximization step.

i	Pre-Max F1	F1	Prec.	Recall	TP	FP	FN
0	64.73%	77.82%	68.50%	92.81%	1059	487	82
1	68.12%	80.00%	73.29%	88.06%	1136	414	154
2	70.01%	84.06%	80.01%	88.46%	1150	286	150
3	71.55%	79.16%	77.13%	81.29%	995	295	229

TABLE I. Performance statistics for the classifier with maximal F1 score for each iteration after the maximization step as well as the original F1 score pre-maximization.

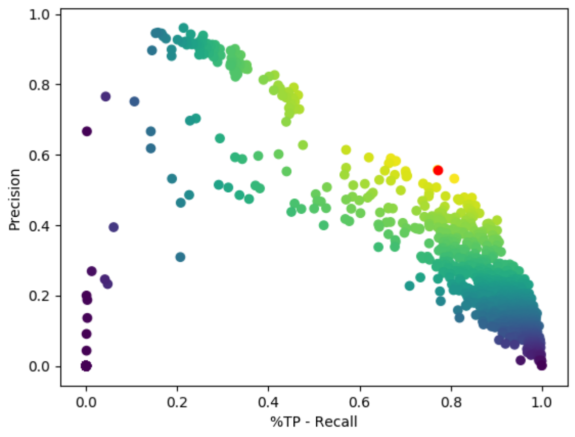

FIG. 1. Precision -vs- Recall for Iteration 0 Pre-Maximization

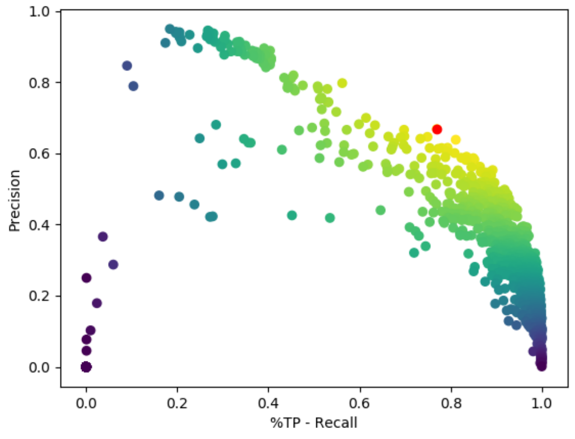

FIG. 2. Precision -vs- Recall for Iteration 3 Pre-Maximization

It is prudent to note that it is possible a different classification parameter set would have had a larger F1 score, in each iteration after the maximization step was conducted, if all misclassified words were checked; however,

i	Starting Size	FP Added	FN Removed	Final Size
0	3691	801	294	4468
1	4468	547	511	4504
2	4504	471	557	4418

TABLE II. The volume of change, in words, of the positive class label dataset label dataset for each iteration.

i	Food	Gardening	Ecosystems	Cumulative	Total FP
2	40.21%	37.06%	6.64%	84.3%	286
3	52.54%	18.64%	5.08%	80.0%	295

TABLE III. Percentage of FP misclassifications of Sister Class words for iterations 2 and 3 for the maximum F1 score classifier after the maximization step.

conducting the label correction on all misclassifications of a classifier is much to costly in time. This is most likely the reason that the post-maximization F1 scores do not clearly show an increase over iterations, while the pre-maximization F1 scores do.

Throughout the maximization step of each iteration a pattern clearly presented itself. Subjectively, very many words that were misclassified seemed like they were comming from several very similar but different, sister, semantic subjects. For the last two iterations, a dataset was created to aggregate positive class labels for those subjects. Upon analysis of the composition of the *plant* classes false positive misclassifications, it was found that on average 82% of FP misclassifications were of actually due to words from three sister semantic subjects: ”food”, ”gardening”, and ”ecosystems”. Table III displays the percentage of words 'sister' subject misclassifications for the top classifier of iterations two and three, as recorded in table I.

Classification was ultimately conducted only with NN and RF classifiers. SVM and KNN classification was untenable due to the size of the data set, where the data was on the order of 3 million rows with 300 dimensions each. For example, the SVM classifier was given over 60 hours of computation time and still had not completed one classification. Interestingly, the neural network was found to preform best at this task[4], for all iterations, by a large margin. For example, in the last iteration the best F1 score for the NN was 0.715534, the best F1 score for RF was 0.631271. If this does not seem like a lot, consider that there were 82 parameter combinations for the neural network that resulted in better F1 scores than the random forest in this same iteration.

VII. FUTURE WORK

There are several things that can be done to build upon this work. This includes both further increasing the performance of the classifiers, making the maximization step even less supervised, and applying the features generated by recognizing the semantic subject of a word to NLP tasks such as text categorization or hierarchical classification of words.

As noted previously, roughly 82% of false positive misclassifications on average consisted of words more closely related to sister semantic subjects. By simultaneously training the classifiers to classify the sister subjects as well, it is very likely that the classifiers would prefer to label the misclassified words according to their proper subject and the false positive rate of the target semantic subject classification will significantly decrease - with a potential reduction of 80%.

When conducting the label checking part of the maximization step in each iteration, a very monotonous procedure is conducted. For most words this procedure consists of a Google search and a subsequent quick scanning of the results to analyze the context that this word is used in. This can be automated by having an algorithm analyze the contents of the Google results and assess whether or not enough words from the target semantic subject exist in the results. If they do, then the algorithm can safely assume that the word is related to the subject and self-correct the labels. This could save a significant amount of time and potentially completely automate the iteration process.

Evaluating the performance of the data captured by this semantic recognition process would be invaluable to proving the utility of it. Both text categorization, for example on the popular Reuters-21578 dataset[13, 26, 44], or the hierarchical classification of words would be excellent areas to apply the feature set learned upon due to the semantic core of the problems.

VIII. CONCLUSIONS

The results demonstrate that an exploratory attempt at recognizing the semantic subject of words by word embeddings can be conducted in an efficient manner for an arbitrary subject and yield very workable results.

Word embeddings can be successfully leveraged to build data sets of words, organized by semantic subject, in an efficient manner. It has also demonstrated that that modern classifiers are capable of generalizing upon this labeled data with reasonable performance and several clear approaches to achieving even better performance.

In conclusion, the process of semantic subject recognition is efficient and tenable. The features produced by semantic subject recognition can be utilized to improve the performance of classifiers in NLP problems much like NER and POS have done in many NLP problems already. This procedure has the potential to be a significant NLP tool and should be further studied.

[1] L. Douglas Baker and Andrew Kachites McCallum. Distributional clustering of words for text classification. *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval. ACM*, 1998.

[2] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL (1)*, pages 238–247, 2014.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[6] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[7] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter*, 6(1):1–6, 2004.

[8] Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 2003.

[9] ACC Coolen. A beginner's guide to the mathematics of neural networks. In *Concepts for Neural Networks*, pages 13–70. Springer, 1998.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[11] Cicero Nogueira dos Santos and Victor Guimarães. Boosting named entity recognition with neural character embeddings. 2015.

[12] Cícero Nogueira dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. *ICML*, 2014.

[13] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. *Proceedings of the seventh international conference on Information and knowledge management*, 1998.

[14] Tang et al. Learning sentiment-specific word embedding for twitter sentiment classification. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics.*

[15] John R. Firth. *A synopsis of linguistic theory 1930–1955.* Oxford:Blackwell, 1957.

[16] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004):41, 2004.

[17] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[18] Nathalie Japkowicz et al. Learning from imbalanced data sets: a comparison of various strategies. In *AAAI workshop on learning from imbalanced data sets*, volume 68, pages 10–15. Menlo Park, CA, 2000.

[19] Thorsten. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, 1998.

[20] Rie Johnson and Tong Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. *Advances in neural information processing systems. ACM*, 2015.

[21] Daniel Jurafsky and James H. Martin. *Speech and Language Processing.* Prentice Hall, Inc., 2014.

[22] Giridhar Kumaran and James Allan. Text classification and named entities for new event detection. *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. ACM*, 2004.

[23] Siwei et al. Lai. Recurrent convolutional neural networks for text classification. *AAAI*, 2015.

[24] Thomas K Landauer and Susan T. Dutnais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review.*

[25] E.D. Liddy. *Natural Language Processing.* Encyclopedia of Library and Information Science. Marcel Decker, Inc., 2001.

[26] Xiao Lou and A. Nur Zincir-Heywood. Combining word based and word co-occurrence based sequence analysis for text categorization. *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, 2004.

[27] Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[28] Chris McCormick. Word2vec tutorial - the skip-gram model. *http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/.*

[29] T. Mikolov. Statistical language models based on neural networks. phd thesis. 2012.

[30] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems.*

[32] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. *HLTNAACL.*

[33] Kazuhiro Morita, El-Sayed Atlam, Masao Fuketra, Kazuhiko Tsuda, Masaki Oono, and Jun ichi Aoe. Word classification and hierarchy using co-occurrence word information. *Information Processing and Management*, 2003.

[34] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticæ Investigationes*, 2007.

[4] Data available at https://github.com/uladkasach/Word-Subject-Classification/tree/v1.0/z_conclusions/

[35] David Nadeau, Peter D. Turney, and Stan Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. *Conference of the Canadian Society for Computational Studies of Intelligence*, 2007.

[36] Manabu Nii, Shigeru Ando, Yutaka Takahashi, Atsuko Uchinuno, and Reiko Sakashita. Feature extraction from nursing-care texts for classification. *Automation Congress, 2008. WAC 2008. World*, 2008.

[37] Alexandre Passos, Vineet Kumar, and Andrew McCallum. Lexicon infused phrase embeddings for named entity resolution. 2014.

[38] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. *EMNLP*.

[39] Ellen Riloff and Wendy Lehnert. Information extraction as a basis for high-precision text classification. *ACM Transactions on Information Systems (TOIS)*, 1994.

[40] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2010.

[41] Beatrice Santorini. Part-of-speech tagging guidelines for the penn treebank project (3rd revision). 1990.

[42] Tobias et al. Schnabel. Evaluation methods for unsupervised word embeddings. *EMNLP*, 2015.

[43] Fabrizio. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 2002.

[44] C. Silva and B. Ribeiro. The importance of stop word removal on recall values in text categorization. *Neural Networks, 2003. Proceedings of the International Joint Conference on*, 2003.

[45] Sharad Singhal and Lance Wu. Training multilayer perceptrons with the extended kalman algorithm. In *Advances in neural information processing systems*, pages 133–140, 1989.

[46] Yanmin Sun, Mohamed S Kamel, Andrew KC Wong, and Yang Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[47] Hirotoshi Taira and Masahiko Haruno. Feature selection in svm text categorization. *AAAI/IAAI*, 1999.

[48] Yong Wang, J. Hodges, and Bo Tang. Classification of web documents using a naive bayes method. *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, 2003.

[49] Peter Wiemer-Hastings, K. Wiemer-Hastings, and A. Graesser. Latent semantic analysis. *Proceedings of the 16th international joint conference on Artificial intelligence.*

[50] Yiming Yang and Xin Liu. A re-examination of text categorization methods. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM*, 1999.

[51] Mohammed J Zaki and Wagner Meira Jr. *Data mining and analysis: fundamental concepts and algorithms.* Cambridge University Press, 2014.

[52] Will Y. Zou, Richard Socher, Daniel Cer, and Christopher D. Manning. Bilingual word embeddings for phrase-based machine translation. *EMNLP*, 2013.