Supervised vs Unsupervised Methods in Word Sense Disambiguation

Sydney Richards
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
rich1143@morris.umn.edu

ABSTRACT

Natural language is full of ambiguity. Many words can have different meanings in different contexts. Word Sense Disambiguation (WSD) is the task of determining the correct meaning of an ambiguous word given its context. Two common approaches to WSD are supervised and unsupervised machine learning methods. These two approaches may also be combined to form a third approach, semi-supervised. Supervised methods typically produce the best results, but these methods require labeled training data that may be difficult and expensive to obtain. Consequently, current research is concentrating on semi-supervised and unsupervised methods. In this paper, we survey a supervised, a semi-supervised, and an unsupervised approach to WSD in order to compare these different approaches.

Keywords

Word Sense Disambiguation, Word Embedding, Shotgun-WSD, It Makes Sense

1. INTRODUCTION

Natural language is full of ambiguity, many words can have different meanings in different contexts [10]. Word Sense Disambiguation (WSD) is the task of identifying which *sense* of an ambiguous word is being used in a given context [4]. A *sense* is a definition or meaning of a word. For example, in the sentence, "I am going to plant an apple tree", the word "plant" has multiple possible senses including:

1. Noun: Living organism of the kingdom Plantae
2. Noun: Industrial plant; Facilities for production
3. Verb: Sow; place seed in ground to grow

In this example, the context of the ambiguous word is "I am going to" and "an apple tree". Using the context, the WSD system must decide which sense of the word "plant" is intended by the author.

WSD is an open problem in the field of Natural Language Processing (NLP), and a solution to this problem would benefit many applications in the field of NLP. WSD can be used as an intermediate step in many NLP applications such as information retrieval, machine translation, content analysis, and word processing. Any situation in which a computer is trying to understand human language, WSD will improve the results. In machine translation, for example, the word "penna" has three distinct senses in Italian. Depending on the context, "penna" can be translated into English as "feather", "pen", or "author" [9]. Therefore performing WSD before performing machine translation will improve machine translation results.

In Section 2 we explain background necessary to understanding the WSD methods. In Sections 3, 4, and 5 we survey a supervised, semi-supervised, and unsupervised approaches to the WSD problem respectively. Lastly, we conclude with a summary of these three methods in Section 6.

2. BACKGROUND

2.1 Supervised Machine Learning

Supervised machine learning algorithms are "supervised" in that they receive labeled training data and go through a training process. Labeled training data consists of inputs labeled with their desired outputs. In the context of WSD, labeled training data consists of ambiguous words in context each labeled with the correct sense of the ambiguous word.

In the training process, the algorithm analyzes the training data and attempts to create a function that maps the inputs to the expected outputs. It will test this function on the training data and compare its output to the expected output. It then modifies its function and tries again. It will continuously run its function on the training data, check its output against the expected output, modify its function, and try again until the training process is complete.

This training process results in algorithms that excel at classifying the words in their training data, therefore their advantage is highly accurate results. However, their disadvantage is that supervised WSD algorithms are only as good as their training data. For example, if the training data does not contain the word "plant" to mean a facility, then the algorithm will not know of this possible sense and therefore cannot successfully classify it in testing. The solution to this is to train the algorithm on more data, but it is expensive and time consuming to do so.

2.2 Unsupervised Machine Learning

Unsupervised machine learning algorithms do not receive labeled training data and may not go through a training process. Unsupervised algorithms may be given resources. For example, an unsupervised WSD algorithm may be given

UMM CSci Senior Seminar Conference, November 2017 Morris, MN.

access to a dictionary. It uses the dictionary to look up the possible senses for a word, then classifies an instance of that word into one of the possible senses. This particular approach is considered a *knowledge-based* unsupervised approach.

Unsupervised algorithms are typically less accurate than supervised algorithms because they do not receive feedback on their output through a training phase. Their advantage is that they do not require large amounts of training data that are expensive and time-consuming to obtain.

2.3 Semi-Supervised Machine Learning

Supervised machine learning algorithms are highly accurate but require large amounts of labeled training data. In contrast, unsupervised machine learning algorithms are less accurate, but require no labeled training data. A compromise between these two approaches is the use of a semi-supervised approach. Semi-supervised machine learning algorithms are similar to supervised algorithms. However they incorporate both labeled and unlabeled data, or data that is labeled in some aspects and unlabeled in others.

2.4 Calculating Word Embeddings

The task of WSD is difficult in part because computers do not understand the meanings of words the way we do. To overcome this, many WSD systems use word representations known as *word embeddings* rather than the typical character representation. A word embedding is a unique mapping of a word to a vector in a multidimensional vector space. This vector space representation of words is highly advantageous for the WSD task because words that are similar, or that might be used in similar contexts, will be close to each other in this space [11]. For example, the word embedding for the word "dog" would be close to the word embedding for the word "pet" because these words are likely to appear in the same context. However the word embedding for the word "dog" would not be close to the word embedding for the word "boat" because these words are less likely to appear in the same context.

2.4.1 Neural Networks

A neural network is a computing system consisting of layers of simple processing nodes. Each node contains an activation function through which it does its processing. Nodes within a network are connected connected to other nodes; an individual node may be connected to many nodes in the previous layer from which it receives data and to many nodes in the subsequent layer to which it sends data. A node assigns weights to each of its incoming connections and, when receiving numbers, it multiplies each by the corresponding weight. It then sums the products to produce a single number. This number is input into the node's activation function. The output of the activation function is sent along all of the node's outgoing connections. The main purpose of the activation function is to allow the neural network to learn complex patterns. Weights are initially set to random values, but as a neural network trains, it continually updates these weights as it tries to find the optimal configuration of weights to complete its task.

A feed-forward neural network (FFNN) is one in which data passes through in the forward direction. These neural networks typically consist of three types of layers: input, hidden, and output layers [8]. FFNNs are common approaches to generating word embeddings. *word2vec* [8] is an algorithm that takes an unlabeled source text as input to generate a word embedding for each word found in that source. Because the input data are not labeled, this is an unsupervised algorithm. word2vec uses one of two FFNN models to generate embeddings: *Continuous Bag-of-Words Model* or *Continuous Skip-gram Model*. [6]

2.4.2 Continuous Bag of Words (CBOW) Model

The CBOW neural network network model is trained to predict a target word after taking in its context words as input. Context words are the words occurring within a certain range before and after the target word in text. This model is a variation on the FFNN described in Section 2.4.1. This model removes the hidden layer and instead consists of input, projection, and output layers. The input layer takes in the context words and encodes each as a vector using *one-hot encoding*. One-hot encoding encodes a word as an n-dimensional vector where n is the number of words in the text, with a 1 in the position corresponding to that word's position in the text, and 0s in every other position. Next, all vectors from the input layer are multiplied by the same weight matrix and sent to the projection layer. The projection layer sums each of these incoming vectors to produce a new vector. The projection layer sends this new vector to the output layer, this represents the neural network's prediction of the target word. This process trains the neural network to recognize words that will be used in similar contexts.

Before training, the weights in the weight matrix are initialized to random values. As the neural network trains, it continually updates these weights. When the training phase is finished, rather than using the neural network for the task it was trained for, the weight matrix is taken as output. Each row of this matrix is a word embedding for the corresponding word in the input text. This neural network is illustrated in Figure 1. In this figure, W(t) denotes the target word, W(t-1) denotes the context word directly to the left of the target word, W(t+1) denotes the context word directly to the right, and so on. [7]

2.4.3 Continuous Skip-gram Model

The Skip-gram neural network model works similarly to the CBOW model. This model is trained to take in a target word and predict words that will appear in the context of that target word. Therefore the input layer takes in only one input word to be encoded as a one-hot vector. This vector is multiplied by the projection matrix and then sent to the output layer. The output layer has a node for each word in the text, the output of each node is the probability that this word will appear in the context of the target word. As in the CBOW model, the weights of the projection matrix are continually updated during training. The weight matrix is the output of the training process, and each row of the matrix is a word embedding for the corresponding word in the input text. This neural network is illustrated in Figure 1. [7]

3. A SUPERVISED APPROACH

A supervised approach to biomedical WSD is presented in [11]. This method uses the Support Vector Machines supervised learning algorithm and incorporates word embeddings from the biomedical domain. Therefore this method may be considered supervised even though the word embeddings were generated in an unsupervised manner.

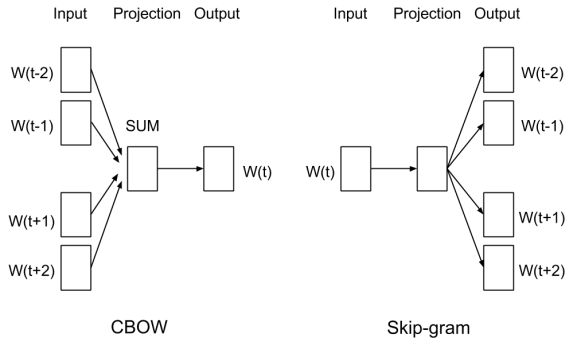

Figure 1: Word embedding generating models [8]

3.1 Aggregation of Word Embeddings

Yepes et al. use word embeddings to train the Support Vector Machines algorithm for the WSD task [11]. The word embeddings were generated using the CBOW approach described in Section 2.4.2. The word embeddings were generated from the 2014 MEDLINE corpus, a collection of written texts containing biomedical information. Next, the word embeddings must be aggregated before being input into the SVM algorithm. This is done in [11] by taking the average of the word embeddings in the context of the target word. The averaged word embedding can be input into the SVM algorithm. This facilitates training and testing the algorithm on different context window sizes. [11]

3.2 Support Vector Machines (SVMs)

Support Vector Machines (SVM) is a supervised learning algorithm. In the training phase, the algorithm takes in labeled training data of two distinct groups. The data consists of vectors in an n-dimensional space, each labeled with which group it belongs to. The SVM algorithm then draws a hyperplane through the space that separates the members of the two groups. It does this by observing the locations of the two groups in the space and draws the hyperplane so that every member of the first group is on one side and every member of the second group is on the other side[1]. The separating hyperplane is the output of the algorithm. In testing, the hyperplane can be used to classify any new vector into one of the groups by observing on which side of the line it lies. [1]

Yepes et al. [11] train an SVM algorithm for the WSD task. The SVM algorithm takes in the word embeddings generated in the previous step as input. When training SVM classifiers for the WSD task, one classifier must be trained to classify each possible sense of each ambiguous word in the training data. Recall from Section 1 that the word "plant" has three possible senses. Assuming these are the only possible senses, three classifiers must be trained in order to correctly classify the word plant. One classifier is trained to distinguish the first sense from the second and third. A second classifier is trained to distinguish the second sense from the first and third. Finally, a third classifier is trained to distinguish the third sense from the first and second. This is repeated for each ambiguous word in the training data. In testing, any new instance of the word "plant" can be classified into its most likely sense of the word "plant" by comparing it to these three classifiers. [11]

[1]In the simplest case, the SVM algorithm can separate groups by a straight hyperplane. We will be assuming this case for the purposes of this paper.

Table 1: Supervised SVM Results

Method	MSH WSD	NLM WSD
SVM Unigrams	93.94%	88.00%
SVM: S 100 W 150	94.31%	-
SVM: S 500 W 50	94.49%	89.63%
SVM: S 300 W 150	94.49%	-
SVM: S 150 W 50	-	90.42%

3.3 Testing and Results

This method was tested on the National Library of Medicine NLM WSD, and the MSH WSD datasets. MSH is a dataset developed using the Unified Medical Language System Metathesaurus and the MEDLINE biomedical database. Both datasets contain ambiguous words and abbreviations from the biomedical domain, the same domain the algorithm was trained on. Yepes et al. [11] report results of experiments using SVMs and word embeddings with different parameter configurations given in Table 1. Parameter S represents the size of the vectors or word embeddings. W represents the window size, or how many context words surrounding the target word are being considered. For example, SVM: S 100 W 150 describes a test of the SVM algorithm using 100-dimensional vectors and considering context windows of 150 words surrounding each ambiguous word. This method's results are compared to SVM Unigrams, a method using SVM with *unigrams* rather than word embeddings. Unigrams are single words extracted from the text.

Table 1 shows scores produced by these methods in testing. Missing entries indicate an experiment was run on only one dataset. The evaluation measure used is *accuracy*, the number of correct answers given over the total number of answers given [9]. This method using SVM with word embeddings achieves highly accurate results. It achieved nearly 95% accuracy on the MSH WSD testing dataset, outperforming the SVM Unigrams method. [11]

4. A SEMI-SUPERVISED APPROACH

One method to overcome the need for a large amount of labeled training data for supervised machine learning methods in WSD is the use of semi-supervised methods which require less training data. Taghipour and Ng [10] describe an approach to WSD using the It Makes Sense (IMS) supervised WSD software. IMS takes in three features: context words, part of speech tags of context words, and local collocations. IMS uses the SVM supervised classifying algorithm.

4.1 Word Embeddings

The semi-supervised approach described in [10] adds word embeddings to the IMS tool as an additional feature in order to preserve more information about the original words. Pre-published word embeddings from [5] were used. The data used to generate these embeddings was Wikipedia and Reuters RCV1, an archive of labeled newswire stories [5]. These datasets are not specific to any domain and are created using an unsupervised algorithm. For these reasons,

this method is considered semi-supervised. These embeddings were trained using an FFNN described in Section 2.4, using the Stochastic Gradient Descent (SGD) training algorithm.

The SGD algorithm randomly selects a window of text from the data and replaces the center word with one chosen randomly from the dictionary. It feeds both the original window and the corrupted window to the neural network. The neural network is trained to assign high scores to original windows and low scores to corrupted ones. The neural networks weights are initialized to random values, and these values are updated as the SGD algorithm trains the network. As in Section 2.4.2, these weights are then taken as the output of the training process and each row in the weight matrix is a word embedding for the corresponding word in the input text. [10]

4.2 Method

IMS is a supervised software tool used for WSD. This tool takes an input text, and extracts three features from it. The first feature extracted is the context words of the target word, all words within a window size of 7 surrounding the target word. The next feature is the part of speech tag of each context word, these are limited to context words within the same sentence as the target word. The last feature is *local collocations*. A local collocation is a word appearing near the target word more often than by chance. IMS finds up to 11 collocations within a window size of 7 surrounding the target word. They are limited to the same sentence as the target word. A fourth feature, word embeddings, are added in [10]. However, word embeddings are much more complex than the software's default features, therefore word embeddings do not fit well into the model. The solution is to scale the word embeddings before adding them to the model. This is done using the below conversion.

$$E = \frac{\sigma E}{stddev(E)}$$

Where σ denotes the desired standard deviation, E the word embeddings matrix, and *stddev* is the standard deviation function that returns a scalar multiple of E. Once these four features are extracted from the input text, they are converted to feature vectors. These feature vectors are used to train SVM classifiers in the same way as described in section 3.2. [10]

4.3 Testing and Results

The algorithm was trained on a subset of the Brown Corpus, a machine readable corpora of American English texts. A training set was created by selecting the top 60% most frequently occurring ambiguous words from this corpus. This method was tested on the Senseval-3 all-words data set. Senseval-3 consists of 3 text documents totaling 2081 ambiguous words. This dataset tests the algorithm on English all-words, that is, they test the algorithm on all ambiguous words in a text, not just words that the algorithm received in training data. If IMS is tested on a word that it has not been trained on, it will assign to that word its first sense from the machine-readable dictionary WordNet [12].

Table 2 shows the scores produced by the IMS tool using word embeddings in testing. Again, the evaluation measure being used is accuracy. Their results are compared both to the IMS method using only the default features, and to WNs1. WNs1 chooses each word's first sense from WordNet, a machine-readable dictionary. The methods using IMS software with word embeddings outperform both the baseline algorithm and the default IMS software, achieving 68.00% accuracy on the all-words dataset. [10]

Table 2: Semi-Supervised IMS Results

Method	Senseval-3
WNs1 baseline	62.40%
IMS	67.60%
IMS + word embeddings	68.00%

5. AN UNSUPERVISED APPROACH

Another method used to overcome the large amounts of labeled training data required by supervised machine learning methods in WSD is the use of unsupervised methods, which require no training data. An unsupervised algorithm, titled ShotgunWSD, based on the Shotgun DNA Sequencing technique is presented in [4]. ShotgunWSD is a knowledge-based unsupervised algorithm. It uses pre-trained word embeddings and accesses WordNet, a machine-readable dictionary. WordNet is organized by *synsets*, sets of synonyms. However strict synsets are not used in the method described in [4], many other relationships between words are included as well such as antonyms and hyponyms. For example, "plant" and "tree" would belong to one synset (living organism) while "plant" and "factory" would belong to another synset (facilities for production). ShotgunWSD is a deterministic algorithm; given the same input document and parameter settings, it will produce the same result each time. [4]

5.1 Sense Embeddings

ShotgunWSD requires a quantitative measure of how related two senses are, this is known as their *relatedness*. One approach to calculating relatedness between senses is to generate word embeddings, convert them to sense embeddings, then measure how related the sense vectors are. Pre-published word embeddings are used by [4]. These embeddings were generated using the word2vec Skip-gram model described in Section 2.4.3 using 3 million words and phrases taken from Google News data as input. This input data is not specific to any domain.

Generating word embeddings results in a cluster of word embeddings for each synset. In the next step, sense embeddings are derived from these clusters of word embeddings. This derivation takes advantage of the reliability of word embeddings to group words by relatedness. In order to compare the relatedness of senses rather than words, a cluster of word embeddings can be combined and generalized into one sense embedding representing the meaning of the synset. This is done by finding the word embedding at the center of the cluster and assigning its meaning to the sense embedding. A simplified example of this is given in Figure 2. Once the sense embeddings are computed, the semantic relatedness between two synsets can be found by computing the similarity between their sense embedding vectors. [4]

5.2 Shotgun DNA Sequencing

The ShotgunWSD algorithm is inspired by the Shotgun DNA sequencing technique [3]. This technique takes a strand of DNA as input, and outputs the most linkely DNA se-

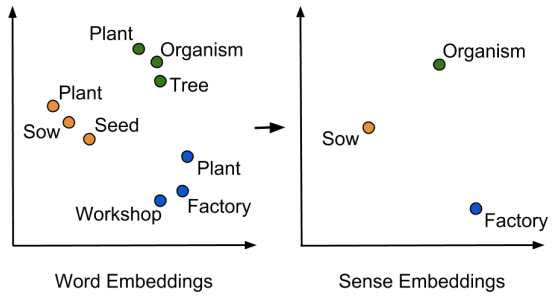

Figure 2: Conversion of word to sense embeddings

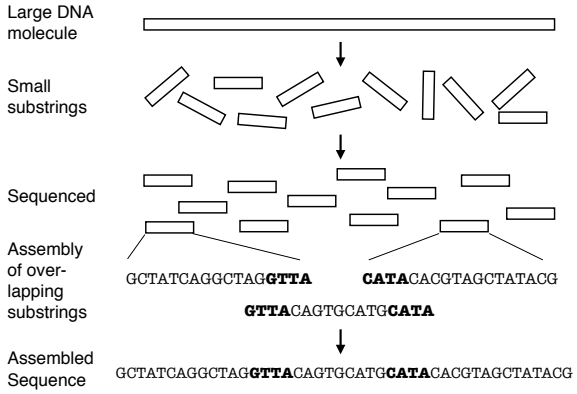

Figure 3: Shotgun DNA Sequencing [2]

quence for that DNA strand. First, copies of the input DNA strand are made, and many sample substrings of a fixed length are taken from these copies. Next, each sample substring is sequenced. If any of these sequences are of low quality or difficult to read, they are typically removed before moving on. Ideally, this should not cause any gaps in the resulting DNA sequence because there are a large number of copies being sequenced. Once each small substring has been sequenced, the substrings are pieced together by merging their overlaps - the longer the better - in order to produce the final DNA sequence. These matches are not perfect, the goal is to find the most likely sequence. This technique is illustrated in Figure 3. [4]

5.3 Algorithm

The ShotgunWSD algorithm follows the same concept as Shotgun DNA sequencing, but with a different goal. The goal of ShotgunWSD is to take a text document as input, and to output a sense configuration for that document that matches the sense configuration a human would produce. Here, the text document to be disambiguated corresponds to the long DNA strand being sequenced, and short context windows within this document correspond to the many short substrings taken from the DNA sequence. Consider the following example sentence.

"I am going to plant an apple tree"

This sentence will serve as our entire document for the purposes of this example. The algorithm begins by selecting one window of up to n words at every possible location in the document, resulting in overlapping context windows covering the entire document. Selecting windows of up to 5 words at every ambiguous word from our example produces the following two context windows.

"going to plant an apple"

"plant an apple tree"

A brute-force algorithm is used to compute all possible sense configurations for each of these windows. Possible senses are chosen from the machine-readable dictionary WordNet. It is important to note that the run-time of the brute force algorithm is exponential with respect to the number of words per window. As a result, the window size parameter n is kept to less than 10. Each of these possible sense configurations is assigned a score based on the semantic relatedness between the word senses within that configuration. This is computed using sense embeddings as described in section 5.1. Below are a few possible sense configurations for our example. For simplicity we assign relatedness scores of 1 for related synsets and 0 for unrelated ones.

1 "going to [living organism] an [fruit]"
0 "going to [factory] an [fruit]"
1 "going to [sow] an [fruit]"
1 "[living organism] an [fruit] tree"
0 "[sow] an [Apple Inc.] tree"
1 "[sow] an [fruit] tree"

In ShotgunDNA sequencing, low quality sequences are typically removed before the assembly phase. Similarly, the lowest scoring sense configurations are removed in ShotgunWSD, and the rest move on to the assembly phase. The number of sense configurations to be kept per window is set by an external parameter c. Below is the set of possible sense configurations after removing the lowest scoring configurations.

1 "going to [living organism] an [fruit]"
1 "going to [sow] an [fruit]"
1 "[living organism] an [fruit] tree"
1 "[sow] an [fruit] tree"

Next is the assembly phase, where possible sense configurations of overlapping context windows are merged if they agree on senses. To do this, the algorithm checks if the suffix of one sense configuration matches the prefix of the next. It begins by considering prefix and suffix lengths of $min(4, n-1)$, and considers shorter context lengths until it reaches 1, then no further merges can be made. Below are possible sense configurations from two consecutive windows, these configurations agree on the sense [living organism] for "plant" and [fuit] for "apple". These configurations will be merged based on these overlaps.

"going to [living organism] an [fruit]"
"[living organism] an [fruit] tree"

When merging two sense configurations, the relatedness score of the resulting configuration is computed using the scores of the configurations being merged. Below are the results of merging our possible sense configurations into larger possible sense configurations. For simplicity, the new sense relatedness scores are computed by adding the existing scores.

2 "going to [living organism] an [fruit] tree"
2 "going to [sow] an [fruit] tree"

When the assembly phase is complete, the algorithm begins assigning senses to each word in the document. The possible sense configurations for each word in the document are ranked by length. Because sense configurations

Table 3: ShotgunWSD Results

Data set	ShotgunWSD	MCS baseline
SemEval2007	79.68%	78.89%
Senseval-2	57.55%	60.10%
Senseval-3	59.82%	62.30%

are merged based on their agreement between senses, longer configurations imply more agreement on a particular sense. Longer configurations are therefore more likely to choose the correct sense. Finally, the sense of each word is chosen by observing the predominant sense from the top k ranked sense configurations containing that word. In the example above, there are two possible sense configurations to decide between. This illustrates the fact that larger window sizes and longer sense configurations typically produce better results. For example, the sense [sow] may be correctly chosen if the surrounding context mentions "shovel". [4]

5.4 Testing and Results

ShotgunWSD was tested on the SemEval2007, Senseval-2, and Senseval-3 datasets. These datasets consist of a total of 11 text documents containing 6,823 ambiguous words. Each tests the algorithm on English all-words data, that is, they are not specific to any domain. ShotgunWSD does not go through a training phase before testing. Its results are compared to the Most Common Sense (MCS) baseline. This is an algorithm that assigns to each word its most frequently used sense. This is determined based on human annotation of the datasets.

ShotgunWSD's three external parameters were consistent for each of the tests. The largest context window that allows the algorithm to run in a reasonable amount of time is 8 words. Therefore n was set to choose context windows of 8 words. c was set to keep 20 sense configurations per context window. This gives plenty of sense configuration options without using excessive time and space. Finally, k was set to choose top the 15 sense configurations per word from which the final sense is chosen. The evaluation measure used is an $F1$ score, the weighted harmonic mean of P and R. Where P is *precision*, the number of true positives over the number of true positives and true negatives. R is *recall*, the number of true positives over the number of true positives and false negatives.

$$F1 = \frac{2PR}{P+R}$$

Table 3 shows scores produced by ShotgunWSD in testing. The table shows that ShotgunWSD only outperforms the baseline algorithm on the SemEval2007 dataset producing its highest score of 79.68%. [4]

6. CONCLUSIONS

In this paper we surveyed three approaches to Word Sense Disambiguation. First, a supervised machine learning approach. This approach is considered to be supervised because it generates word embeddings from domain-specific biomedical data, then trains the SVM supervised learning algorithm on this data. This method produced results of almost 95% accuracy and outperformed the baseline algorithm when tested on biomedical test data, the same domain it was trained on.

Second, a semi-supervised machine learning approach. It uses the supervised IMS WSD software. This approach incorporates word embeddings generated in an unsupervised manner from data that is not domain-specific, then trains the SVM supervised learning algorithm on this data, therefore it is considered semi-supervised. This method produced results of 68% accuracy and outperformed the baseline algorithm when tested on all-words test data, that is, data that is not specific to any domain.

Third, an unsupervised machine learning approach using an algorithm based on Shotgun DNA sequencing and sense embeddings. This approach is unsupervised because it was not trained on any labeled training data. This method produced results with F1 scores of 60 - 78% when tested on all-words data. These results do not show much improvement over the baseline algorithm.

Acknowledgments

Thank you to Elena Machkasova, Skatje Myers, Kristin Lamberty, and Peter Dolan for their guidance and support.

7. REFERENCES

[1] Introduction to support vector machines. *OpenCV 2.4.13.3 documentation*.

[2] Whole genome sequencing. *Genetics Generation*.

[3] S. Anderson. Shotgun DNA sequencing using cloned DNase i-generated fragments. *Nucleic Acids Research*, 9(13):3015–3027, 1981.

[4] A. M. Butnaru, R. T. Ionescu, and F. Hristea. Shotgunwsd: An unsupervised algorithm for global word sense disambiguation inspired by DNA sequencing. *CoRR*, abs/1707.08084, 2017.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011.

[6] L. Hardesty | MIT News Office. Explained: Neural networks, Apr 2017.

[7] C. McCormick. Word2vec tutorial - the skip-gram model, Apr 2016.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[9] R. Navigli. Word sense disambiguation: A survey. *ACM Comput. Surv.*, 41(2):10:1–10:69, Feb. 2009.

[10] K. Taghipour and H. T. Ng. Semi-supervised word sense disambiguation using word embeddings in general and specific domains. In *HLT-NAACL*, pages 314–323, 2015.

[11] A. J. Yepes. Word embeddings and recurrent neural networks based on long-short term memory nodes in supervised biomedical word sense disambiguation. *Journal of Biomedical Informatics*, 73(Supplement C):137 – 147, 2017.

[12] Z. Zhong and H. T. Ng. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of the ACL 2010 System Demonstrations*, ACLDemos '10, pages 78–83, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.