A COMPARATIVE ANALYSIS OF WORD EMBEDDINGS FOR STANCE DETECTION

HAMSA HUSSEIN

Abstract. This paper approaches the shared task of Semeval 2016 task 6 subtask a, stance detection in tweets in a simple way assessing the perfomances of 4 different word embeddings across 3 different machine learning algorithms. Some models prove to have preferences when it comes to word embeddings while others didn't. Across all 12 models, the highest accuracy we obtained is f1 score of 59

Introduction

Stance detection, the task of detecting whether a text is in Favor, Against, or neutral towards a specific target, has become increasingly important in many domains, with politics being a major one. The SemEval 2016 task 6 subtask A dataset, Stance detection in tweets, provides a great benchmark for this research.

In our research, we use two sets of datasets, training and testing datasets, with both being provided by SemEval 2016 task 6 subtask a. Each dataset contains tweets labelled with target (5 distinct targets) and stance (favor, aginst or none). Our task is to train a machine learning model and test it against the testing dataset.

We will firstly turn the texts into vectors in which we will use 4 word embeddings and use 3 different machine learning techniques, which in total will result in 12 models. The main purpose of this is to experiment with which word embeddings work best for a given a machine learning algorithm.

1. Related Work

Since this dataset comes under a shared task, there is one major paper (Mohammad et al. 2016) that explains the task, dataset and their reserach on it. It explains how they collected the data and how they approached the task.

While the main paper used SVM with n-grams and unigrams which is machine learning techniques, there were other approaches that used deep learning methods. Vijayaraghavan, et al. 2016 approached it with CNN method and Zarrella, et al. 2016 approached with RNN and LSTM methods.

Our approach is quite different compared to relative research work on this dataset. Given a tweet, it could be in favor or against depending on the given target, e.g. "Donald Trump should win" is in favor of Donald Trump but against Hilary Clinton. Hence every research paper's approach was to assess their model for stance detection based on a given target, i.e. they split the datasets based on targets. However, our approach differs on that since we do not split the data based on targets, one major reason being such tweets are rare.

Instead, what we focus on is given a machine learning technique, which word embedding is best suited for it. So we use 4 words embeddings against 3 different machine learning techniques and train each model on the training data and assess it on the testing data.

Elfardy, et al. 2016 seems to have a similar approach to ours. They were testing lexical semantic features. It was more focus on what to feed the ML model than which model to use. They also used SVM classifier.

One ML technique that appears in all papers is SVM, which seems to be the best ML technique that can be used in stance detection hence we included it in one of our three ML methods.

2. Methods

In our methodology for stance detection, we used three different models to test their performance across 4 different word embeddings.

We used 4 different word embeddings, Glove, Word2Vec Skip-gram, Word2Vec CBOW (Continuous bag of words), and TF-IDF.

The 3 models we used to assess these word embeddings are Perceptron classifier, Naive Bayes classification, and Support Vector Machine (SVM).

We calculated the accuracy (micro f1-score) as well as the weighted f1-score for each model.

3. Experiment

The dataset we are using is composed of 2814 tweets for training and 1249 tweets for testing. For each tweet, we have its target and stance: FAVOR, AGAINST, NONE.

3.1. **Preprocessing.** We first read the training data named "Semeval.txt", which wasn't encoded in uft8. So we use chardet package to detect the right encoding and load the data. After that we preprocess the data by changing all the text into a lower case, removing mentions, hashtags punctuations and stopwords.

We then name the columns; we set the tweet column as "Tweet" and stance column as "Stance".

We do the same for the test data, named as $Test\_data$, with tweet and stance columns denoted as $Tweet\_test$ and $Stance\_test$.

3.2. **Word embeddings.**

3.2.1. *Glove.* We used the 6B tokens with 300 dimension. We then defined an empty dictionary *glove_embedding* to store the word-vector pairs. We then defined a function, *get_sentence_embedding_Glove*, to retrieve the vector representation of the tweets. We provided an example to see how it works with the first tweet of the training data.

3.2.2. *Skip-gram.* We used the function Word2Vec from the package gensim.models in which we can set the variable sg to 1 for skip-gram. We setted *vector_size* as 300 (same as Glove). We then defined a function to retrieve the vector given a word. We also printed an example.

3.2.3. *Continuous bag of words.* We just changed the sg variable from the Word2Vec function to 0.

3.2.4. *TF-IDF.* We used unigram+bigram ($ngram\_range$ = (1,2)) for our TF-IDF word embedding so that we can capture key words (unigrams), e.g. "support" and "oppose", and 2 phrase words (bigrams), e.g. "highly recommend".

3.3. **Model.**

	Accuracy	Weighted F1-score
Glove	0.26	0.23
Skip-gram	0.57	0.42
CBOW	0.24	0.10
TF-IDF	0.34	0.36

TABLE 1. Results for Perceptron model

	Accuracy	Weighted F1-score
Glove	0.51	0.53
Skip-gram	0.31	0.31
CBOW	0.36	0.37
TF-IDF	0.60	0.49

TABLE 2. Results for Naive Bayes classification model

	Accuracy	Weighted F1-score
Glove	0.59	0.58
Skip-gram	0.57	0.42
CBOW	0.57	0.42
TF-IDF	0.59	0.58

TABLE 3. Results for SVM model

3.3.1. *Perceptron classifier.* We trained our model in these following steps

(1) Get the 4 word embeddings for the tweet column and store it in their respective variables.
(2) Convert those variables into PyTorch tensor (using PyTorch package).
(3) Define the perceptron model.
(4) Initialise the model (with input dimension = 300 for Glove, Skip-gram and CBOW)
(5) Define the optimiser (using SGD)
(6) Train the model (for loop and loss.backward() function)

We then evaluated the accuracy of the model as well as the weighted f1 - score in which you can see in table x.

3.3.2. *Naive Bayes Classification.* We use the multinomialNB function for training TF-IDF and the GuassianNB function for training the other word embeddings, both imported from *sklearn.naive_bayes*. We then made predictions and evaluated the model through $classification\_report$ and calculating the accuracy as well as the F1-score.

3.3.3. *SVM.* We imported SVC from the sklearn.svm package and trained 4 models, one for each word embedding. We used a linear kernel, which means that the model tries to find a linear decision boundary between classes. We tried non-linear kernels, but more on this later in the "discussion" section. We then evaluated the model by doing the same, classification report and f1-score.

3.4. **Results.** The results we obtained are in Tables 1, 2 and 3.

4. Discussion

As we can see, the tables show the accuracy of the model and the weighted f1-score. We choose the accuracy because it is equal to the micro f1-score but rounded to 2 decimal places and weighted f1-score because our dataset had class imbalance.

Based on the tables, Naive Bayes classification and SVM scored the highest accuracy using TF-IDF as word embedding, while Perceptron model scored the highest accuracy with Skip-Gram. However, there is one issue we faced with the perceptron model, which we will discuss later to highlight which word embedding is better to use for that particular model.

For Naive Bayes classification model, we can see that the accuracy between the model using TF-IDF and Glove (2nd best) is nearly 10%. One reason for that is because Multinomial Naive Bayes (MultinomialNB) assumes that each feature (word) is independent of others given the class label, which TF-IDF happens to be aligning with that (it provides sparse, independent-like representations). Also, MultinomialNB works best with Count-Based features which is what TF-IDF is.

For SVM, we can see the numbers are much closer to each other compared to Naive Bayes classification model. One reason for this is because SVM does not care if the input (word embedding) is sparse (TF-IDF) or dense (Glove, Skip-gram and CBOW), as long as the decision boundary can still be drawn effectively.

4.1. **Problems.** For Perceptron model, we faced one major problem during training it. We first trained the model without adjusting the weights of the classes, however that resulted in the model only predicting for one class, e.g. "FAVOR" for everything. So we had an accuracy level of around $15-25\%$ for all of the word embeddings. So we adjusted the weights and even tried different loss function (adam). However, some of the model still was predicting for just one class (results in table 4).

Earlier, upon seeing the results table, we thought perceptron model had the highest accuracy using Skip-gram as a word embedding, however based on table 4 results, we can see it's not as good of a model compared to Glove and TF-IDF because it predicted for one class for all values. Considering both tables, one could argue TF-IDF and Glove work best for perceptron model as a word embedding compared to Word2Vec.

We come across this problem as well in SVM model, which both CBOW and Skip-gram predicted only for "AGAINST".

4.2. **Improvements.** We had the most trouble with Perceptron model. The biggest contributor to higher accuracy was adjusting the weights. Increasing the number of epochs and changing the optimiser function to adam did not contribute much.

For TF-IDF, we used unigrams + bigrams ($ngram\_range = (1, 2)$) instead of unigrams which boosted the accuracy of the perceptron model significantly and slightly in the other models.

For SVM, one thing to note is that when we choose the kernel, we chose linear. We did experiment with non-linear kernel (Kerneal = "rb"), but the accuracy of the model did not change significantly hence why we kept it as linear.

Overall, the best model has an accuracy of 59%. One major improvement we could do adjusting the sizes of the training and testing data. Usually, a dataset is split into 80% training and 20% testing, however in our case, we didn't split the data but used a new set of data as a test data with their ratio being 65% to 35% (training : test).

	AGAINST	FAVOR	NONE
Glove	117	579	553
Skip-gram	1249	0	0
CBOW	0	1249	0
TF-IDF	492	250	507

TABLE 4. Counter for perceptron model

5. CONCLUSION

Based on our analysis, we could agree that word2vec wasn't great across all models in general compared to the other word embeddings. Whereas TF-IDF seems to be the best word embedding in general across all models as it scored the highest accuracy in all models.

REFERENCES

[1] MOHAMMAD, S. M., KIRITCHENKO, S., SOBHANI, P., ZHU, X., & CHERRY, C. (2016). *SemEval-2016 Task 6: Detecting Stance in Tweets*. In Proceedings of SemEval-2016 (pp. 31–41). Association for Computational Linguistics.

[2] VIJAYARAGHAVAN, P., SYSOEV, I., VOSOUGHI, S., & ROY, D. (2016). *DeepStance at SemEval-2016 Task 6: Detecting Stance in Tweets Using Character and Word-Level CNNs*. MIT Media Lab, Massachusetts Institute of Technology Cambridge, MA 02139.

[3] ZARRELLA, G., & MARSH, A. (2016). *MITRE at SemEval-2016 Task 6: Transfer Learning for Stance Detection*. The MITRE Corporation.

[4] ELFARDY, H., & DIAB, M. (2016). *CU-GWU Perspective at SemEval-2016 Task 6: Ideological Stance Detection in Informal Text*. In Proceedings of SemEval-2016 (pp. 434–439). Association for Computational Linguistics.

APPENDIX A. DATASET AND CODE

This link contains both py and ipynb for my code as well as the both datasets I used, training and testing dataset.
https://github.com/husseinh6/NLP-assignment-1

COVENTRY UNIVERSITY
Email address: husseinh6@uni.coventry.ac.uk