CS447 Final Project Report: Twitter Sentiment Analysis with Linear SVM and Logistic Regressor

Yong Jin Kim[1]

[1]University of Illinois Urbana-Champaign
ykim164@illinois.edu

Abstract

This final project report describes how I gradually developed my understanding of text sentiment classification by implementing an SVM classifier and a Logistic Regressor and performing some further experiments to seek approaches that also work. The main system that I tried to implement for the final project is the Multi-view Ensemble approach by NILC-USP that participated in SemEval 2017. NILC-USP uses three classifiers which is trained in a different feature space using bag-of-words model and word embeddings to represent sentences and uses a Linear SVM and a Logistic Regressor as base classifiers. Instead of implementing the voting ensemble, I focused more on how each of three classifiers was built, strengths and drawbacks of the system, and how well it performs.

1. Introduction

The rise of social network services has vastly accelerated the formation of people's opinions and thoughts about all existing topics in the world. Due to this rapid increase in the speed of content generation, observation of people's opinions from social network services has become a new trend of comprehending and learning the public trend or public sentiment. Among all social network services, Twitter has been the most popular medium for collecting thoughts and opinions due to its accessibility and boundless range of topics covered by numerous users. (Rosenthal et al., 2017)

Studies of analyzing the opinions and sentiments of a Twitter post have not been very successful due to the lack of dataset until a shared task of Twitter sentiment analysis was brought up by International Workshop on Semantic Evaluation (SemEval) in 2013. Starting from 2013, SemEval has been running the task for four consecutive conferences providing usable and well-formatted datasets retrieved from actual Twitter posts. Those subtasks that have been covered in the conference include topic-based polarity classification, 5-point scale Twitter sentiment classification, etc.

Despite the increased interest and attention to the task of Twitter sentiment analysis, it still remains as a difficult task since the length of a single post is limited and the posts include a high rate of nonstandard language. A significant amount of efforts have been made to tackle these problems in previous conferences; some of them include simple unsupervised models that use precompiled sentiment lexicons for evaluating polarity of tweets and some sophisticated supervised models that use feature representations in combination with machine learning algorithms such as Support Vector Machines or deep neural networks.

As a student who recently became intrigued by the power and effectiveness of Machine Learning, I wanted to take this final project as an opportunity to experience and feel how well-built Machine Learning model can be used to solve difficult tasks like Twitter sentiment analysis. To do this, I read some of the reports that describe the system used in the SemEval 2017 for the Twitter sentiment analysis task and picked several papers that used Support Vector Machine (SVM) for solving the task. I chose SVM because I had an intuition that SVM would perform really well on Twitter posts where numerous features can be extracted from. Also, it is a system that could be built promptly after some procedures of preprocessing of Twitter text data, so it serves my purpose of my final project which is doing multiple experiments. (Rosenthal et al., 2017)

The main system that I tried to implement for the final project is the Multi-view Ensemble approach by NILC-USP that participated in Semeval 2017.

This final project report describes how I gradually developed my understanding of text classification by implementing an SVM classifier and performing some further experiments to see if some approaches also work.

2. Background

Two most common approaches used in tackling this task were deep learning and supervised Support Vector Machine (SVM). There were at least 20 teams who used deep learning and neural networks such as CNN and LSTM networks while SVMs were also popular. The top 5 teams for the Message Polarity Classification task in English used deep learning and three of the top 10 scoring teams used SVM classifiers with various features extracted from the text data.

Ensemble methods, which are skills that generate multiple models and then combine them to produce maximum performance also stood out. One related work that I did not implement but was successful in the conference was TakeLab's SVM classifier with a rich set of features. Ranking 5th place on the two-point scale English Tweet classification task, TakeLab's system shows how effective an SVM becomes when it is combined with various features extracted from a tweet data. (Lozic et al., 2017)

TakeLab extracted various features from tweets including the standard Tf-idf feature, pre-trained word embeddings, real-world ratings of a given topic, and its own special measure called nostalgia feature. Most features were extracted from websites that store real-world ratings for various domains including movies, TV shows, actors, games, musicians, companies, etc. These real-world features are what make TakeLab's system especially powerful; one of the reasons that make SVM a powerful classifier is that it uses a real-valued feature so that it can retrieve the result accurately with more realistic assumptions.

One unique feature that they adopted is the feature called "Nostalgia Feature", This feature is extracted based on the assumption that a topic from long years ago tends to be mentioned in a positive way relative to some new topics due to nostalgia. This is very interesting because it shows that the way how human normally think can be applied to some tasks that are difficult to solve.

One thing that this work could have improved was expanding the domain of the topic for rating retrieval. For example, they did not retrieve the ratings for sports athletes which is one of the most frequently mentioned group not only on Twitter but in most Social Network Services. By analyzing TakeLab's system gave me a good insight of how carefully features have to be selected when using an SVM and creative selection of features like the nostalgia feature used in TakeLab's system could also contribute to the performance of a system.

With these understandings, I decided to implement another system from SemEval2017 that used an SVM which is implemented by NILC-USP with less number of features than TakeLab's system. I chose this system to better understand the effect of some of the most commonly used features like Tf-idf and word embeddings. Moreover, I chose a system that is opened for improvement (or did not perform best in competition) to experiment with my own features after I completely implement the original model.

3. Task

The motivation of the Twitter Sentiment Analysis task lies on its potential value in the real world. The wealth of information that Twitter posts contain can be used as a powerful measure for improving the quality of analysis or solving difficult tasks that cannot be solvable in a normal approach. These are why this task is worth challenging.

The Twitter Sentiment Analysis task proposed by SemEval include five subtasks for English which are classified based on the number of scales used for evaluation (five-point scale ranges from strongly-positive to strongly-negative). Some topics just take a tweet as an input and ask the system to determine the sentiment of the tweet itself while some topics also provide a specific topic along with a tweet and ask for the sentiment conveyed towards that particular topic. The task that I mainly focused on is the Subtask A which asks the system to determine the sentiment of the tweet in three-scale (negative, neutral, and positive). The dataset of the task includes Twitter posts annotated for sentiment on a 2, 3, and 5-point scales. The tweets were selected based on recently trending events and the topics included a wide range of popular figures and issued events.

The dataset that I used was the Twitter2016-test data which was provided in SemEval2017 and contains 20632 tweets in total.

The primary evaluation measure for the Subtask A is the average recall, which is recall averaged across the positive(P), negative(N), and neutral(U) classes. The average rec is computed as follows:

$$AvgRec = \frac{1}{3}(R^P + R^N + R^U)$$

where R^P, R^N, and R^U refer to the recall of positive, negative, and neutral class respectively. Another evaluation measure is accuracy which is straightforward.

4. Implementation

As I mentioned, the system that I implemented for the final project is the multi-view ensemble approach for Twitter Sentiment Analysis proposed by NILC-USP from University of Sao Paulo (USP). Ensemble in Machine Learning is a technique that creates multiple models and combines them to retrieve the best result. It usually produces solutions more accurately than a single model. NILC-USP's system particularly used a voting ensemble approach where each classifier is trained in a different feature space. (Correa et al., 2017)

The first step of the voting ensemble is to create multiple classification and regression models using the dataset. Each base model can be created using different splits of the same training dataset and same algorithm, or using the same dataset with a different algorithm.

The system uses three different feature spaces to be used in the voting ensemble. The first feature space that the system uses is the bag-of-words model with a Linear SVM. The second and third feature spaces are Linear SVN and Logistic Regressor that use two different methods of combining word embeddings to represent a single tweet. Instead of implementing the ensemble itself, I implemented all systems that were used in the ensemble to analyze how each system works and to learn the strength and weakness of the system.

4.1 Preprocessing

The ways that the system preprocessed the Tweets are not mentioned in the paper, so I followed a general preprocessing step that include tokenizing and handling some peculiarity of tweets. I used the NLTK Tokenizer library to do this task. NLTK has two tokenizers that are suitable for preprocessing these kinds of data: WordTokenize and TweetTokenizer. The name TweetTokenizer itself indicates that it is a tokenizer suitable for tweets, so I did a simple comparison between two.

Given a sample tweet,

"dear @Microsoft the newOoffice for Mac is great and all, but no Lync update? C'mon"

With the usage of the WordTokenize, the given tweet is tokenized as follows:

['dear', '@', 'Microsoft', 'the', 'newOoffice', 'for', 'Mac', 'is', 'great', 'and', 'all', ',', 'but', 'no', 'Lync', 'update', '?', 'C'mon', '.']

It shows that it recognizes '@' and 'Microsoft' separately which is supposed to be combined and considered as a mention for another account.

Using the TweetTokenizer, the tweet was tokenized as:

['dear', '@Microsoft', 'the', 'newOoffice', 'for', 'Mac', 'is', 'great', 'and', 'all', ',', 'but', 'no', 'Lync', 'update', '?', 'C'mon', '.']

It shows that it recognizes '@Microsoft' as a single token. I also found out that the TweetTokenizer recognizes URLs and hashtags as single tokens. This is an ideal option for preprocessing tweets. However, the system does not take mentions and hashtags into account because the task is not a topic-based classification and mentions and hashtags will not actually contribute to the sentiment of the tweet. So, I decided to stick with the TweetTokenizer instead of WordTokenize since it is easier to drop mentions, URLs, and mentions (check if the first part of the word includes a notation that indicates that the text is one of them and drop), but I decided to leave this option for further experiments with features after getting the initial result.

After that I removed stop words, dropped punctuations, and set all text to lowercase. After all these steps, the final form of a tweet that will be used as our single input would look like:

['dear', 'newooffice', 'mac', 'great', 'lync', 'update', "c'mon"]

4.2 Linear SVM with Tf-Idf

After preprocessing all given tweets, I implemented the first classifier which is the Linear SVM using bag-of-words features weighted by term frequency-inverse document frequency (Tf-idf).

Tf-idf is useful in that it eliminates the limitations that a regular word-count feature, which just takes word frequency into account without considering the sentiment of the sentence, by diminishing the weight of the terms that occur in all the documents and increases the weight of the rare terms.

There were two main steps in this implementation: extracting tf-idf features and training the classifier. For these two steps, I used the Tf-idf Vectorizer and Linear SVC from the Scikit-learn library.

I first had to reformat the pre-processed tweet data to a list of strings where each string represents a single (preprocessed) tweet, because this is the format accepted by Scikit-learn's Tf-idf Vectorizer.

After that, I created label vectors that contain the target for each tweet. I labeled 'negative' sentiment as 0, 'neutral' sentiment as 1, and 'positive' sentiment as 2.

Then, I split the tweet data into 8:2 ratio where 8 is the training set and 2 is the testing set. Both training set and testing set were transformed into Tf-idf vectors, and the transformed training data were trained based on the training label using the Scikit-learn Linear SVC.

Finally, the performance of the trained model was checked by counting how many untrained tweets were labeled correctly using the testing label which is 20% of the label.

As a result, 2396 tweets out of 4126 tweets (58.07%) were labeled correctly.

4.2.1 Experiments on Linear SVM with Tf-Idf

In order to see how the accuracy changes based on how the data is validated, I trained this data again using the cross-validation technique, specifically the K-cross fold validation technique. The K-cross fold validation technique randomly partitions the dataset into k equal parts where each subpart is used for testing while the remaining parts are used for training. It is repeated k times and each subpart is used exactly once for testing. Finally, the result from each iteration can be averaged to produce the best estimation. With an expectation to retrieve higher accuracy, I used the 10-cross-fold validation where 1/10 of the dataset was used for testing the model while the other 9/10 of the dataset were used for training. As a result, 12348 tweets out of 20632 tweets (59.85%) were labeled correctly.

It shows that the use of K-cross fold validation technique slightly increases the accuracy, showing that using a different kind of model validation technique could be one way to improve the performance of the model.

4.3 Linear SVM with Word Embeddings

In word embeddings, each word is represented as a dense, real-valued vector. These vectors are learned by neural network techniques. The intuition of word embedding is that the features of each word are contained by the vector, so it can be said that each value of the vector is a meaningful representation of the word. NILC-USP's system adopted an approach that represents each tweet by averaging the word embedding vectors of the words that appear in a single tweet. In other words, it is an element-wise average that is the word vectors for the individual words in the tweet are summed together and divided by the number of words.

To implement this part, I used the pre-trained vectors trained on Google New dataset which contains 300-dimensional vectors for about 3 million words and phrases. It is normal to use a pre-trained model since it takes too long to train a model with a huge corpus.

I loaded the model in Python by using the Gensim package, which is a Python library for vector space modeling and topic modeling. This time, unlike the Tf-idf modeling where I had to make each tweet to have a single string, the tweet data has to be tokenized into lists of strings. So, I reverted the data back to its initial preprocessed form.

Then, I computed the average of the word embedding vectors of the words that each tweet has by using Numpy's element-wise vector addition function (numpy.add) and dividing the vector by the length of the tweet.

I assume that the reason for this low increase rate is that there were many tweets that contain words that are not in the word2vec dictionary. This was actually expected since the language used in Twitter is mostly informal, including slang and abbreviations. For example, 'lol' is a word that is likely to be used in Twitter, but unlikely to be in the word2vec unless the model is trained on a Twitter-specific corpus.

4.3.1 Experiment with word2vec

To find a way to see if the approach of this system was an ideal approach, I did an experiment with Gensimś similarByVector function that takes a word2vec vector as an input and returns similar words based on the input vector.

Given the following tweet that is labeled as positive:

'Beat Demon Souls. If there is a black friday sale I will pick up dark souls 2, otherwise might just get Dark Souls 1'

Inputting the tweets vector representation (average of word embedding vectors of the tweet) on the similarByVector gives following outputs:

```
[('souls', 0.6736515164375305),
 ('dark', 0.5599076747894287),
 ('Tonight_Vanish', 0.5494640469551086),
 ('Colbert_SpongeBob', 0.5463873147964478),
 ('sweetest_gentlest_kind', 0.5457371473312378),
 ('Carpe_diem_seize', 0.5420875549316406),
 ('Blood_curdling', 0.5310376882553101),
 ('hateful_spiteful', 0.529852569103241),
 ('hellbound', 0.5296430587768555),
 ('soul', 0.5268893241882324)]
```

Figure 1. Similar words of the average word embeddings vector

With the given output, it is hard to see the result as a representation of the tweet since there is no much information that tells that the tweet is positive. Words like 'dark' or 'hellbound' are something that seems more like a negative word. The model predicted this wrong labeling it as a neural tweet.

This shows that getting the average of word embedding vectors of the words that compose the tweet might not be the best approach for this task.

To improve this model, training the model on a corpus that contains Twitter-specific language could also help since it would lessen the number of missing keys in the word2vec dictionary.

4.4 Experiment with word2vec

The third classifier is a logistic regression model that uses weighted word embeddings as the feature. Logistic regression is a linear classifier that predicts the class probabilities for binary classification problems. When there are more than two classes to predict, it uses a one-vs-rest approach. For the implementation, Scikit-learn's LogisticRegression library was used.

For the feature vector, word embedding vectors were used again, but this time the vector was weighted, or multiplied, by the Tf-idf of the word it represents. The intuition behind weighting the word embeddings by the Tf-idf is to reflect how important a word is to a document in the document vector.

To weight the average word embedding vectors for each tweet, I had to get the Tf-idf value for each word in a single tweet and multiply this by the word embedding vector of the corresponding word before getting the final average vector.

To make this process easier, I stored the Tf-idf score for each word in a tweet in a list and multiplied the score by the word embedding vector when computing the average vector of a tweet.

The result of the model prediction was 2380 tweets out of 4126 tweets (57.68%) showing the lowest accuracy of the three classifiers, and the average recall was 52.76

I also used a SVM Classifier with the feature used with the Logistic classifier, and the result of the model prediction was 2377 tweets out of 4126 tweets (57.61%) showing almost no change, and the average recall was 52.01

This shows that changing the classifier does not actually change the result. Thus, this indicates that choosing the weighted word embeddings feature was not that ideal. The possible reason for the low performance could be the missing Tf-idf score for some words.

4.5 Further Experiments with Additional Feature Vector

To seek the effect of an additional feature vector, I tested the Linear SVM with Tf-idf feature vector with an additional feature vector called curse word vector. As its name indicates, it is a vector that contains a binary value that shows whether the tweet contains a curse word or not. I was inspired by my thought that all tweets with curse word are very unlikely to be positive. With this intuition, I thought it would be able to contribute to the accuracy of the model.

I first downloaded a list of curse word provided by Google and stored it as a set in my code. Then I created a new vector called curse-vector and added binary values 0, for indicating that there are no curse words in the tweet, and 1 for indicating the usage of curse words. After concatenating it to the Tf-idf feature vector, I tested the performance the same way I did for all models.

As a result, the model predicted 2380 tweets out of 4126 tweets (58.17%) correctly and average recall of 57.91%. Only four more tweets were labeled correctly compared to the model without the curse word feature.

This result is actually not a surprise because it was hard for a 1-dimensional feature vector to give an impact to the model because the Tf-idf feature vector was already huge. Also, the tweet data was not that straightforward to be determined by an inclusion of curse words; the sentiment of most tweets was determined by some other factors that are not curse words.

Despite the trivial effect of the feature addition, adding creative features could definitely improve the prediction of the model. Tf-idf and word embeddings do not contain all information needed to classify a sentiment of a tweet. As shown in the system submitted by TakeLab, the team that ranked 5th place in subtask B by using an SVM classifier with rich set of features, wise choice of features can highly improve the performance of the model.

5. Result and Conclusion

The result of my implementation of the three classifiers proposed by NILC-USP and two additional classifiers that I experimented with for the Twitter Sentiment Classification on a 3-point scale is as follows:

Table 1: Test Results

	AvgRec	Accuracy
SVM /w Tf-idf	0.579	0.581
SVM /w Tf-idf and Curse Word	0.579	0.582
SVM /w Word Embeddings	0.548	0.600
LogisticReg /w Weighted Word2Vec	0.577	0.528
SVM /w Weighted Word2Vec	0.576	0.520

Since NILC-USP achieved an average recall of 0.612 and accuracy of 0.617 it can be said that my implementations of the systems introduced in the paper were pretty close to the actual system. The fact that the system ranked 20th out of 37 teams shows that it has some drawbacks in the model and could be improved.

Some problems that this system had was that the word embeddings vector were not that useful because it did not contain many of the words in the dictionary due to the informality of Twitter language. It did not recognize abbreviated words like u or r which indicate you and are. This issue can be improved by training the word2vec model in a corpus that contains Twitter specific language.

Another suggestion that I would like to make for the system is to add a name entity tagger to recognize a mention of a person (probably a celebrity) in a tweet and use the data that numerically represents the person (rating if he or she is a movie star) as a feature vector. This way, there will be more information for the classifier since the high rating of the actor indicates the higher probability of him or her being mentioned in positive tweets. Overall, the system does a decent job of classifying sentiments of tweets using concise and straightforward approach.

6. Personal Thoughts

Personally, this project was a great opportunity for me to learn some well-known and frequently-used Machine Learning techniques. This is the first time that I actually implemented a system that solves a Natural Language Processing task. I felt like implementing an actual system is the best way to learn how the algorithm works and how the NLP concepts are applied to the problem-solving strategy. Although it was just imitating the system that was proposed by another person, doing this project gave me a new insight on how to approach these kinds of tasks efficiently and creatively.

References

Edilson A. Correa Jr., Vanessa Queiroz Marinho, Leandro Borges dos Santos. NILC-USP at SemEval-2017 Task 4: A Multi-view Ensemble for Twitter Sentiment Analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation. Vancouver, Canada, SemEval*

Sara Rosenthal, Noura Farra, Preslav Nakov. 2017. SemEval-2017 Task 4: Sentiment Analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation. Vancouver, Canada, SemEval*

David Lozic, Doria Šari ´ c, Ivan Toki ´ c, Zoran Medi ´c, Jan Šnajder . 2017. TakeLab at SemEval-2017 Task 4: Recent Deaths and the Power of Nostalgia in Sentiment Analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation. Vancouver, Canada, SemEval*