UNIVERSITY OF NORTH TEXAS

MACHINE LEARNING PROJECT REPORT

Course Code: CSCE 5215

Semester: Fall 2016

Krishna Chaitanya Sanagavarapu, Dinakar Sadineni, Himanish Kopalle

Project Report on

Sentiment Analysis on Twitter Data set

Sentiment Analysis on Twitter Data set

Krishna Chaitanya Sanagavarapu, Dinakar Sadineni, Himanish Kopalle

Abstract

Sentiment analysis is a process of identifying a person's emotion/attitude towards a particular topic. In this work, we develop models to automatically classify a tweet into positive or negative. We train different classifiers using a twitter data set and compare their performance. The classifiers compared are support vector machines and Naive Bayes over the same data set. This report describes the task, data and results.

1 Task

Sentiment analysis is an important research on understanding and learning public/users opinion which helps in various verticals, such as:

- Determine marketing strategy
- Improve campaign success
- Improve product messaging
- Improve customer service
- Generate leads

Social Media is a rich source of text that portrays a persons attitude/emotion/opinion. Text pre-processing is a major challenge in performing sentiment analysis of social media data. Huge amount of labeled data is also required to automate sentiment analysis. Previous work Jadav and Vaghela (2016) classifies social media text to positive and negative using Support Vector Machines and Naive Bayes. We follow a similar approach to pre-process tweets and train models to classify them into positive or negative tweets

2 Data

The Twitter Sentiment Analysis Data set consisting of data from Sanders (2011) and UMICH SI650 - Sentiment Classification competition contains 1,578,627 classified tweets, each row is marked as 1 for positive sentiment and 0 for negative sentiment. We had huge amount of data available. In our experiments we found that it is time taking to train any single classifier over the entire data. We hence took a subset of this data and reported results.

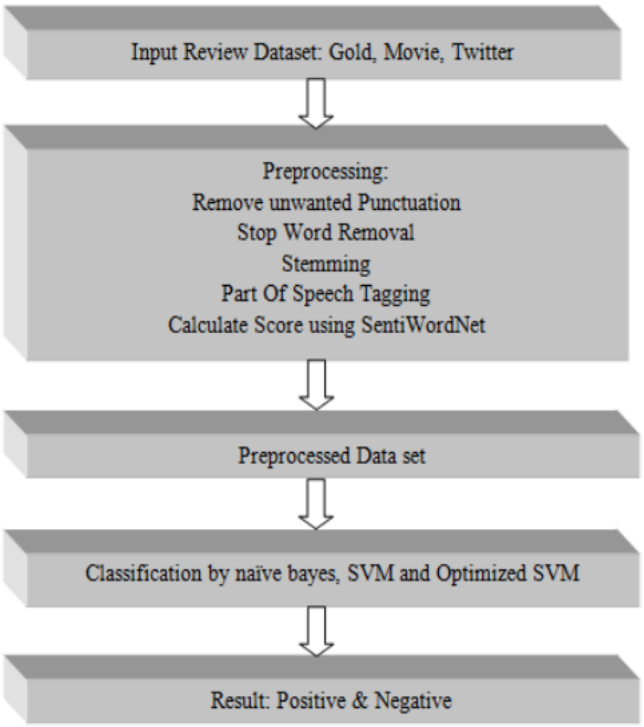

Figure 1: Method

3 Previous Work

The method followed in Jadav and Vaghela (2016) is illustrated in figure 1. We follow similar steps to pre-process tweets before training our models.

In our experiments we do the following pre-processing:

1. Remove unwanted punctuation: All punctuation which are not necessary, it has been removed
2. Stop Word Removal: Some words used more and more time such words are called stop word. This pronouns, prepositions, conjunctions have no specific meaning.
3. Stemming : It converts word into its grammatical root form. Stemming technique converts word like “teach”,”teacher”,”teaching”,”teached”,”teaches” to root word teach. Among many available algorithms, we have used Porter stemming algorithm

4 Implementation

In this section we describe the method used for feature extraction and how we trained different models.

4.1 Feature Extraction

We used tf–idf, short for term frequency–inverse document frequency, a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often

used as a weighting factor in information retrieval and text mining. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

In a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms. In order to re-weight the count features into floating point values suitable for usage by a classifier it is very common to use the tf–idf transform. Tf means term-frequency while tf–idf means term-frequency times inverse document-frequency:

$$tf - idf(t, d) = tf(t, d)idf(t).$$

Using the TfidfTransformer‘s default settings,

$$TfidfTransformer(norm =' l2', use_{idf} = True, smooth_{idf} = True, sublinear_{tf} = False)$$

the term frequency, the number of times a term occurs in a given document, is multiplied with idf component, which is computed as

$$idf(t) = log(1 + n_d(1 + df(d, t))) + 1,$$

where n_d is the total number of documents, and df(d,t) is the number of documents that contain term t. The resulting tf-idf vectors are then normalized by the Euclidean norm:

$$v_{norm} = v(||v||_2) = v({v_1}^2) + {v_2}^2 + \ldots + {v_n}^2.$$

This was originally a term weighting scheme developed for information retrieval (as a ranking function for search engines results) that has also found good use in document classification and clustering.

4.2 Learning

In this project we perform the following experiments:

1. Experiment 1:
    - We select a random 10000 tweets and split it into 6000 training tweets and 4000 tweets for testing
    - We then train an SVM over the 6000 tweets. We optimize the SVM by tuning kernel, C and gamma parameter over 10 fold cross validation on training data.
    - We also train a Multinomial Naive Bayes classifier with the 6000 training tweets
    - We finally report the precision,recall and F1-score over the 4000 test tweets
2. Experiment 2:
    - From above experiment we learned that Naive Bayes classifier is faster
    - We now split considered 50000 random tweets from the data and divided them into 30,000 train and 20,000 test tweets.
    - We report the results of a Multinomial Naive Bayes classifier trained over the 30,000 tweets, and testing them on 20,000 test tweets

We utilized scikit learn Pedregosa et al. (2011) a python library to perform these experiments.

5 Results and Discussion

Table 1 presents results with Experiment 1. The training of an SVM along with its optimization took over 1 minute when 6000 tweets are used for training. This was the case with both processed and un-processed tweets. Naive Bayes classifier was much faster and took only 0.004 seconds to train.

From Experiment 1, we realized training a Naive Bayes classifier is much faster. From the results:

- With Naive Bayes classifier, The overall performance with processed tweets is slightly better compared to unprocessed tweets
- However, training an SVM with unprocessed tweets yielded better results.

Table 2 Presents results with Experiment 2. The training of a Naive Bayes classifier took 0.018 seconds with 30,000 tweets and 0.0014 seconds to test on 20,000 tweets.

From Experiment 2 results:

- With Naive Bayes classifier, The overall performance with processed tweets is slightly lower compared to unprocessed tweets
- Increasing the training data evidently improved the performance of Naive Bayes classifier

Table 1: Results experiment 1: 10 fold cross validation over 6000 random training tweets and tested over 4000 random tweets from the entire corpus

Model	**Class**	**Processed Tweets**			**Un-processed Tweets**		
		Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Optimized SVM	**Positive**	0.71	0.70	0.70	0.74	0.70	0.72
	Negative	0.80	0.81	0.80	0.80	0.84	0.82
	Weighted Avg.	0.76	0.76	0.76	0.78	0.78	0.78
Naive Bayes	**Positive**	0.82	0.46	0.59	0.85	0.42	0.57
	Negative	0.72	0.93	0.81	0.71	0.95	0.81
	Weighted Avg.	0.76	0.74	0.72	0.77	0.74	0.71

Table 2: Results Experiment 2: Naive Bayes trained over 30,000 random tweets and tested on 20,000 tweets

Model	**Class**	**Processed Tweets**			**Unprocessed Tweets**		
		Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Naive Bayes	**Positive**	0.74	0.79	0.77	0.75	0.80	0.78
	Negative	0.74	0.68	0.71	0.75	0.69	0.72
	Weighted Avg.	0.74	0.74	0.74	0.75	0.75	0.75

6 Conclusion

From the various experiments done in this project, we observed that sentiment analysis of tweets require high amount of training data. It is computationally time taking to train any single classifier even with that available data. An optimized SVM performed with an F1 score of 0.78 when trained with 6000 unprocessed tweets.

We compared Naive Bayes and optimized SVM classifiers with respect to their performance and training time. We realized that Naive Bayes is faster to train but portrayed less performance compared to an optimized SVM.

7 Appendix

The following are submitted along with this report:

1. sentiment analysis.py: This python script takes as input the data file and number of tweets to output results with various classifiers
2. Data.csv: A comma separated value file that has tweets and sentiment denoted as positive and negative using 1 and 0 respectively
3. Readme.txt: This file describes the requirements to execute the learning script
4. Experiment 1 and 2 result output files are also submitted

References

Jadav, B. M. and V. B. Vaghela (2016). Sentiment analysis using support vector machine based on feature selection and semantic analysis. *International Journal of Computer Applications 146*(13).

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research 12*(Oct), 2825–2830.

Sanders, N. J. (2011). Sanders-twitter sentiment corpus. *Sanders Analytics LLC*.