Opinion Mining for Text Data

Comparing various feature extraction methods.

Problem:

The problem in this paper pertains to solving the problem of extracting an opinion/sentiment from a huge corpus of data. A major part of the data in the world is text and most of it is ambiguous. Sentiment analysis is an important aspect pertaining to customer service/ social network recommendations. Opinion mining is used by various government corporations to consider the public's response to various policies, and also by various corporations to get an idea of the people's response for a given product. The main problem in sentiment analysis is the decision of selecting the best feature for the model to train. Twitter is an amazing platform as companies, politicians and celebrities have their official accounts on the social networking platform, enabling a direct interaction between the two parties. The tweets have a limit of 140/280 characters making the tweets short and precise, reducing the chances of spam. This article considers supervised learning methods uses a supervised learning method with different extraction techniques. Sentiments considered for this project are either negative or positive. There have been a lot of different approaches to this problem, but it is quite difficult because there are a lot of factors to considered for extracting an opinion/sentiment from a given text.

Dataset:

The dataset used is a pre labelled dataset from kaggle[1], It contain 1.6 million tweets, 800,000 tweets labelled positive, and the other half labelled as negative. The dataset contains six columns: Target(Sentiment):- A value of 0 to 4, where 0 is a negative tweet and 4 is labelled as positive; id: An id associated with the tweet; date: The date of the tweet; Flag: The query used for the tweet, NO_QUERY if there is no query; User: The user who tweeted the tweet; Text: The actual text of the tweet. The size of the data is around 1 Gigabyte. Twitter being a social network sites, there are images(tiny urls), links to other tweets/webpages, usernames and various other non text material that has to be cleaned before a feature extraction technique can be used on the data. There are no outliers or there is no missing data in the dataset.

Sentiment	Tweets
0	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
0	is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!
0	@Kenichan I dived many times for the ball. Managed to save 50% The rest go out of bounds
0	my whole body feels itchy and like its on fire
0	@nationwideclass no, it's not behaving at all. i'm mad. why am i here? because I can't see you all over there.

Table 1. Dataset without preprocessing.

1. Kaggle 140SentimentDataset

Cleaning and Preprocessing:
The columns containing the metadata were removed and just the tweets and the sentiment are cleaned for further processing. All URL and HTML tags are removed. For cleaning, I have used the BeautifulSoup library. Words that have repeated letters like "Yesssssss" used as a colloquial are converted to the basic version ('yes'). Secondly, the hashtags are processed by removing the '#' signs. The @usernames are removed using Regular expressions(Regex). Lastly, multiple spaces are converted to a single space and the text is stripped of all extra whitespace characters.

Sentiment	Tweets
0	aw thats a bummer you shoulda got david carr of third day to do it d
0	is upset that he cant update his facebook by texting it and might cry as a result school today also blah
0	i dived many times for the ball managed to save the rest go out of bounds
0	my whole body feels itchy and like its on fire
0	no its not behaving at all im mad why am i here because i cant see you all over there

Table 2. Dataset after preprocessing.

Strategies:

The main aim of this paper is to select various feature extraction techniques, and apply them to some simple machine learning algorithms. The impact of these techniques are considered and compared. The approach starts with using a simple countvectorizer, that counts the number of times the word occurs in the corpus. As the data is balanced, choosing a countvectorizer is not a bad option. The second technique is using a more accepted approach of taking the Term Frequency- Inverse Document Frequency (TF-IDF) of all the terms dividing it by the number of documents the word occurs in. The formula for TFIDF is :

$$\mathrm{wf}_{t,d} = \begin{cases} 1 + \log \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}.$$

$$\text{wf-idf}_{t,d} = \mathrm{wf}_{t,d} \times \mathrm{idf}_t.$$

TFIDF:

TFIDF is a weight used in text mining. It assigns weights to a word denoting the importance of the word. The importance of the word proportionally increases to the number of times a word appears in the document, but it is divided by the number of documents that contain the word. This offset occurs because if a word occurs in many documents, it cannot be used to differentiate between those documents, reducing its importance in some sense.

Term Frequency: (No. of times term t appears/number of total terms in the tweets)
Inverse Document Frequency: $\log_e$(Total number of tweets/ No. of tweets with term t)

For TFIDF, I used sublinear TF scaling. Sublinear TF scaling assumes that a word, even if it occurs thirty times in a document, would not have a weight that is thirty times the weight of a single occurrence term. Hence, for TFIDF calculation, logarithm of the term frequency is used instead of the term frequency itself.

Word2Vec:

The third approach uses a two layer deep learning neural network library called word2vec which uses a large dataset of corpus and generates word embeddings. Word2vec is a neural network implementation that learns distributed representations for words. The library reads all the sentences in the corpus, and considers the usage of the words in the corpus. The model then plots the words in a 300 dimension vector space according to its usage in the corpus. The embeddings by the model were generated from from the dataset itself containing 40,000 sentences. The words that are located nearby have similar meanings. The word2vec does not require labels to create meaningful embeddings. As the words in the various tweets are of varying lengths, the vectors generated are normalized with the number of words in the tweet. The stop words are not removed as the word2vec considers the whole context of the sentence for word distributions.

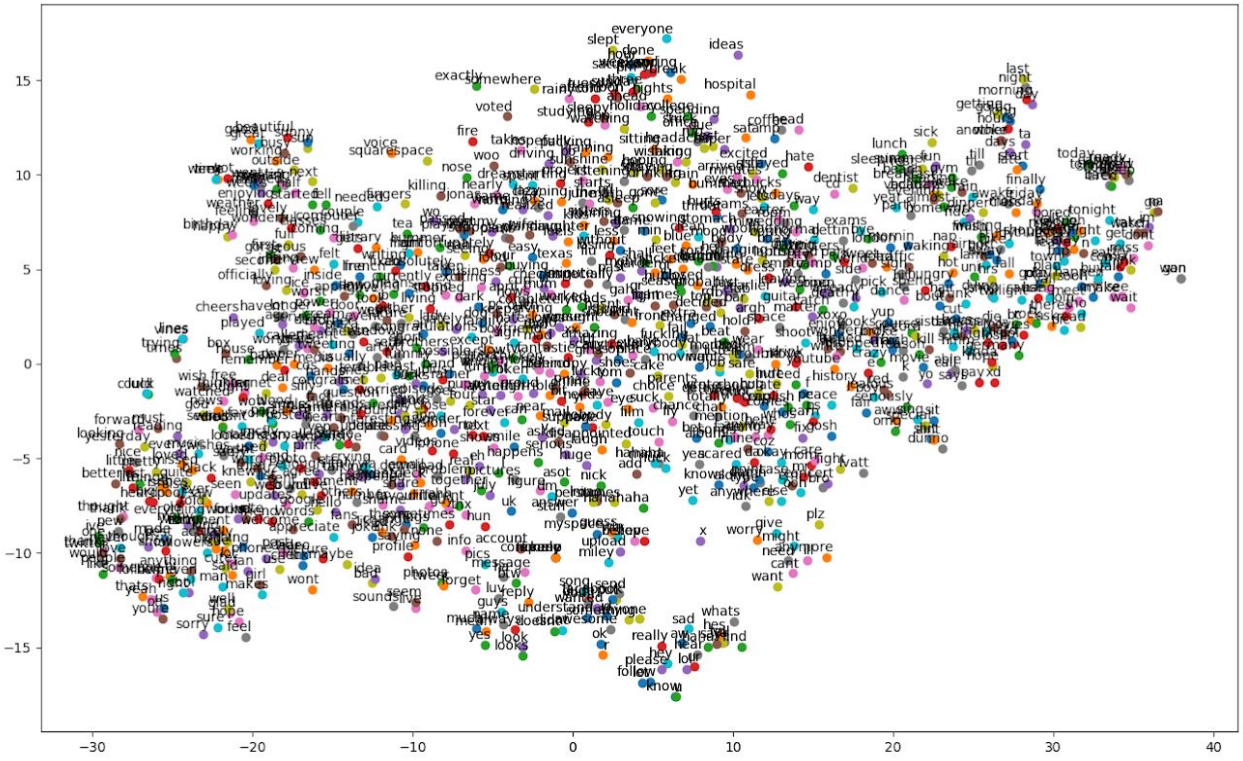

Fig 1. Word2Vec embeddings in 300 dimension space.

The paper uses Naive Bayes, Support Vector Machine (SVM), Random Forest Classifier, Logistic Regression and K-NN algorithm. Multinomial NB and Linear SVC versions of the respective models are used for this project.

Machine Learning Algorithms:

Naive Bayes:
Naive Bayes is a very robust and fast classifier for text mining. Multinomial NB is suitable for discrete values and works well with the TFIDF that's why it is used for this problem.

$C^* = argmac_c P_{NB}(cjd)$

$$P_{NB}(c|d) := \frac{(P(c)\sum_{i=1}^{m} P(f|c)^{n_i(d)})}{P(d)}$$

Here, f represents the feature after the extraction and $n_i(d)$ is the count of the feature in the tweet. There are total m features. C is the class that is assigned to a tweet d. Here, P(c) and p(f\c) is calculated.

Logistic Regression:
Logistic regression is used when the dependent variable is a set of binary numbers and not a cardinal set of numbers. As, we have 0 as negative and 1 as positive, we can use logistic regression for training our model. Logistic regression mainly uses sigmoid function as the underlying function. It gives the probability of the the label class, using the features.

P(sentiment = true | tweet) = implies the probability that the sentiment is true, based on the text in the tweet.

Logistic regression is a linear method, but the classification is done using a log function. The estimation from the training data is done using the maximum likelihood estimation. Another parameter to be considered is C; C defines a float whose value defines the regularization strength.

SVM:
SVM is a non probabilistic binary linear classifier. It is representation of points in a space, mapped in such a way that there is a clear space between the points belonging to two different labels. The new data is mapped in the same space and are predicted where they will belong. They are very important and strong model for text classification. The paper uses Linear SVC version of the sklearn's SVM.

Random Forest:
It is an ensemble learning classifier. Random Forest outputs mode of the classes decided by a multitude of decision trees. They are used as they are a correction over the overfitting problem of decision trees. There is a very minute difference between the results of gini index and entropy, and so, gini index is used as a function to measure the quality of the split. The number of trees in the forest are considered 100.

Methods:

The dataset uses KFold cross validation (K = 10).

1) The text part of the dataset is cleaned using the preprocessing function before transforming the text using a TFIDF vectorizer. The words for each tweet are given a term frequency score which is normalized with the number of documents. The TFIDF vectorizer returns a sparse matrix containing the number of document, the id of the word and its TFIDF score. The feature is used with Naive Bayes, SVM, Logistic Regression, K- NN and Random Forest Classifier.

2) TFIDF is used as a feature extraction without preprocessing the data. This is done to gauge the impact of text cleaning for accurate classification of text data. The feature is used with Naive Bayes, SVM, Logistic Regression, K- NN and Random Forest Classifier. 1 - 3 ngrams are considered without removing the stepwords.

3) Instead of dividing the term frequency with the document frequency, a CountVectorizer is used which just considers the number of times a word occurs in the text. The feature is used with Naive Bayes, SVM, Logistic Regression, K- NN and Random Forest Classifier.

4) Word2Vec is used as a feature extraction method. A model is trained using the complete dataset and word distributions are plotted in a multi dimensional space. The training set vectors and test set vectors are developed using the model's vocabulary. Every word in the dataset is parsed, and if it has a feature vector, the feature vector is added to that word. As the number of tweets have different lengths, the vectors are normalized before fitting them to a model. The models used are SVM, Logistic Regression and Random Forest. The number of fimensions are taken to be 300.

5) Using Latent Semantic Analysis (SVD - Singular Value Decomposition) along with TFIDF to reduce the dimensions to apply Word2Vec without running out of memory, as well as selecting the best features out the current set of features.

Evaulations:

As the data here is balanced (Equal number of classes), so just the accuracy score would be enough to compare the impact of various feature extraction techniques and the machine learning models used.

1) TFIDF with Text pre processing:

Models	Accuracy score	f1score	precision	recall	true negative	true positive
Naive Bayes	0.7787	0.767636	0.808121	0.731176	1652	1462
Logistic Regression	0.776775	0.780095	0.7687823	0.791953286	1523	1583
K-NN	0.51335	0.521397	0.513029	0.530668	992	1061
SVM	**0.7869**	**0.789855**	**0.779148**	**0.801031**	**1545**	**1601**

Table 2: Scores for TFIDF with PreProcessing.

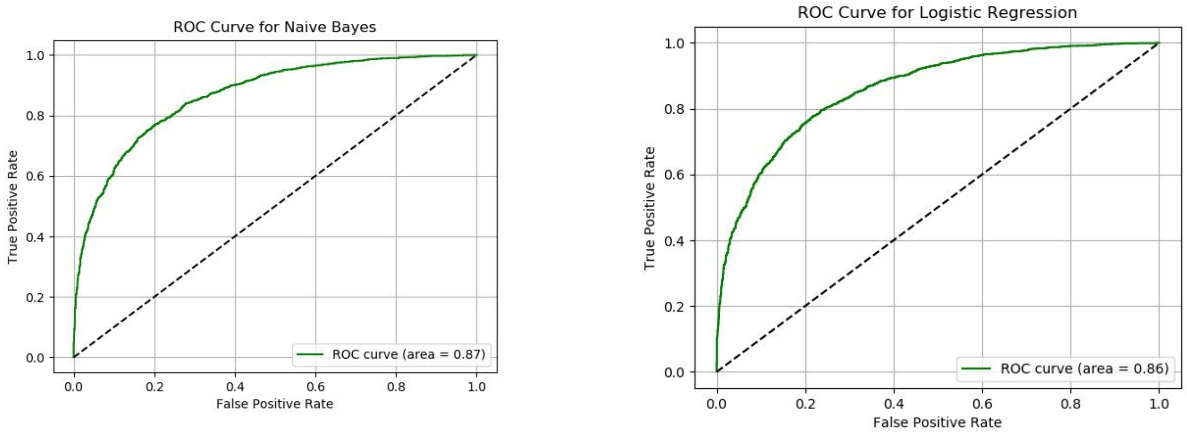

Fig. 2 ROC curves for Naive Bayes and Logistic Regression

2) CountVectorizer with preprocessing:

Models	Accuracy score	f1score	precision	recall	true negative	true positive
Naive Bayes	0.77355	0.7706931	0.780427	0.76131	1571	1522
Logistic Regression	**0.7856**	**0.7877442**	**0.780012**	**0.795772**	**1550**	**1591**
K-NN	0.609525	0.6282273	0.59945	0.660427	1117	1320
SVM	0.772175	0.7748820	0.765837	0.784229	1520	1568

Table 3: Scores for CountVectorizer with PreProcessing

As the dataset is balanced, and the tweets are collected using words of varying topics, the accuracy score for countvectorizer and the TFIDF with basic text processing without stemming gives a very similar score (+- 0.01%). SVM (Linear SVC) for TFIDF and Logistic Regression for Countvectorizer are the most promising of the models.

3) TFIDF without Text Preprocessing:

Models	Accuracy score	f1score	precision	recall	true negative	true positive
Naive Bayes	0.771175	0.7556320	0.810952	0.707794975	1669	1415
Logistic Regression	0.7588	0.7617100	0.752787	0.771211394	1493	1542
K-NN	0.682125	0.6302321	0.752725	0.543466738	1641	1087
SVM	**0.7778**	**0.7806044**	**0.770861**	**0.790824366**	**1529**	**1581**

Table 4: Scores for TFIDF without PreProcessing

Without preprocessing, i.e. removing the @, # symbols, tags, repeated letters in the words, and simply using TFIDF, a decrease of 1.5 - 2 % in the accuracy score is seen. Also, the classification of the negative label (sentiment score = 0) is seen to be more accurate and it decreases for the positive label (sentiment score = 1).

4) Word2Vec:

Models	Accuracy score	f1score	precision	recall	true negative	true positive
Logistic Regression	0.537375	0.548383	0.538075	0.559094	2052	2247
Random Forest	**0.53775**	**0.524556**	**0.542698**	**0.507589**	**2262**	**2040**
SVM	0.527625	0.58785	0.523301	0.670565	1526	2695

Table 5: Scores for Word2Vec with PreProcessing

For word2vec, a 80-20% split of the data is used, as the word2vec is a very resource consuming to consider a 10 fold cross validation. Using only Word2Vec and using it with traditional machine learning algorithms, the accuracy scores decrease significantly. One reason could be the size of the data used to train the model. Even with a huge increase in the training for word embeddings, the accuracy score does not see a significant increase. The dimensions

5) LSA for dimension reduction Word2vec:

Models	Accuracy score	f1score	precision	recall	true negative	true positive
Logistic Regression	**0.5175**	**0.541568**	**0.516071**	**0.569715**	**1860**	**2280**
Random Forest	0.50375	0.496065	0.504128	0.488256	2076	1954
SVM	0.50375	0.061909	0.569565	0.032734	3899	131

Using LSA to reduce the dimensions, and selecting the best features (300 dimensions to 2 dimensions) the accuracy score decreases even further. This can be used for further feature extractions as the time taken for processing decreases.

Pros and Cons:

Cons:

1) The actual dataset being too large, various robust and intensive algorithms had to be dropped from consideration.
2) Word2Vec does not work efficiently as clustering / unsupervised learning would give better results.
3) The strategies rely too much on the ground truth and hence, require a labelled dataset for training.

Pros:

1) The results are improvements from some existing solutions which used just TFIDF along with ngrams as a feature extraction.
2) As there are multiple feature extraction techniques and their comparisons, their impact can be studied, and they can be extended.
3) Getting an accuracy of around 80% without using high computing resources (deep learning) for smaller problems.

Conclusion and Future Aspects:

The purpose of this paper was to implement and compare some feature extraction methods for text classification as feature extraction is the most important aspect of machine learning, sometimes, even more than the algorithm selected. For a limited length dataset, Logistic regression and Naive Bayes work the best in almost all cases of the feature extraction method selected. In the future, I would like to work on Word2vec and extend it using unsupervised learning to get some better results as well as some insights in the data. Future work would also include considering neutral tweets as well as text with emojis for sentiment analysis, replicating a much more real world problem. The current dataset is collected using a wide range of topics having no significant relation, and so, in the future, doing some information extraction from the tweets would better help understand the emotion behind the tweet/sentence.