Twitter Sentiment Analysis

Rahul Gupta
rahul20065@iiitd.ac.in

Anjali
anjali20082@iiitd.ac.in

Abstract

In the recent past there has been hike in the usage of twitter as a social media platform where people share opinions about various walks of life.As of May 2020, the average number of tweets sent per minute are around 350,000. In this report we propose a technique for text sentiment classification using term frequency- inverse document frequency (TF-IDF) and compare two different classification models.

1 Introduction

The current decade has become a digital book where each post one shares adds to the sentiment of a particular topic. We are choosing Twitter for sentiment analysis because it is limited to 140 characters of text that's why users can explain their brief ideas via a short message.The sentiment analysis of such tweets can be useful to various manufacturers and marketing personals as they can see by the analysis of the tweets how people who consume their products react to those products.

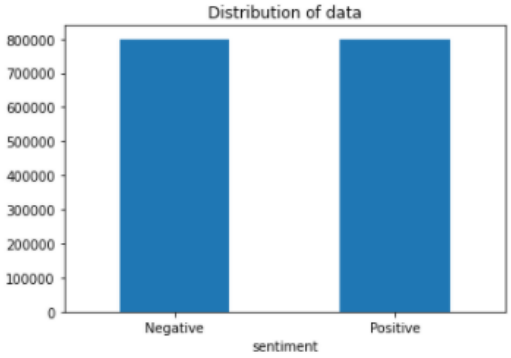
The gradual growth of the number of users and the data related to them, has provided a good impetus to every company or organizations to mine these micro-blogging sites to collect information about people's opinion about their services or products. Due to this increase in user interaction, the future sales of any product or service depends a lot on the sentiments and perceptions of the previous buyers. Therefore, it is necessary to have an efficient way to predict user sentiments about a product or service.

2 Literature Survey or Related Work

This document has been adapted from the instructions for earlier IEEE proceedings : An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation by Bijoyan Das and Sarit Chakraborty and Sentiment Analysis with NLP on Twitter Data by Md. Rakibul Hasan, Maisha Maliha, M. Arifuzzaman. Additional elements were taken from the formatting instructions of the *International Joint Conference on Artificial Intelligence* and the *Conference on Computer Vision and Pattern Recognition*.

3 Dataset

The dataset for this work has been obtained from Kaggle. It initially had 6 columns. But for our analysis only two columns are required- text and sentiment. So we worked on these two features and trained the ML classifier models based on them. Dataset contains the Text data which includes upper lower cases, hyperlinks user references, emojis and non alphabets which need to be either removed or deleted in order to form useful feature from the text data.




```
0          @switchfoot http://twitpic.com/2y1zl - Awww, t...
1          is upset that he can't update his Facebook by ...
2          @Kenichan I dived many times for the ball. Man...
3            my whole body feels itchy and like its on fire
4          @nationwideclass no, it's not behaving at all....
                                ...
1599995    Just woke up. Having no school is the best fee...
1599996    TheWDB.com - Very cool to hear old Walt interv...
1599997    Are you ready for your MoJo Makeover? Ask me f...
1599998    Happy 38th Birthday to my boo of alll time!!! ...
1599999    happy #charitytuesday @theNSPCC @SparksCharity...
Name: text, Length: 1600000, dtype: object
```

4 Data analysis

The text column in the dataset contains:
1. uppercase as well as lowercase data
2. hyperlinks
3. user references
4. emojis
5. some very short words that don't add sentiment to the tweet
6. non-alphabets
The above observations about the data helped us analyse what all data is required for sentiment analysis and which of the above mentioned characteristics are redundant or not useful i.e. do not help in determining the emotion of a tweet.

5 Planned Solution Architecture

This section represents our focus on strategy for sentiment analysis on Twitter data. We first obtain the dataset from an online resource and then we preprocess it.The supervised learning techniques use machine learning on a previously classified dataset which is considered to be almost accurate. These pre-classified datasets are often domain specific, therefore the model it generate can work only for a particular domain. These datasets are first converted into intermediate models where documents are represented as vectors [6], and then the intermediate representations are fed to the machine learning algorithm.The Preprocessing steps taken are:

1.**Lower Casing**: Lowercasing ALL your text data, although commonly overlooked, is one of the simplest and most effective form of text preprocessing. It is applicable to most text mining and NLP problems and can help in cases where your dataset is not very large and significantly helps with consistency of expected output. Each text is converted to lowercase.
2.**Removing URLs**: Links starting with "http" or "https" or "www" are replaced by empty string "". These urls don't add any sentiment in the tweet. So, these are considered noise in the data.
3.**Replacing Emojis**: Replace emojis by using a pre-defined dictionary containing emojis along with their meaning. (eg: ":)" to "EMOJIsmile").
4.**Removing Usernames**: Replace @Usernames with empty string "". (eg: "@xyz" to ""). These usernames don't add any sentiment in the tweet. So, these are considered noise in the data.
5.**Removing Non-Alphabets**: Replacing characters except Digits and Alphabets with a space.These Non-Alphabets don't add any sentiment in the tweet. So, these are considered noise in the data.
6.**Removing Consecutive letters**: 3 or more consecutive letters are replaced by 2 letters. (eg: "Heyyyy" to "Heyy")
7.**Removing Short Words**: Words with length less than 2 are removed.
8.**Removing Stopwords**: Stopwords are the English words which do not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. (eg: "the", "he", "have").For example, in the context of a search system, if your search query is "what is text preprocessing?", you want the search system to focus on surfacing documents that talk about text preprocessing over documents that talk about what is. This can be done by preventing all words from your stop word list from being analyzed.
9.**Lemmatizing**: Lemmatization on the surface is very similar to stemming, where the goal is to remove inflections and map a word to its root form. The only difference is that, lemmatization tries to do it the proper way. It doesn't just chop things off, it actually transforms words to the actual root. For example, the word "better" would map to "good". Lemmatization is the process of converting a word to its base form. (e.g: "Great" to "Good")

After the preprocessing steps, the dataset is fed to machine learning classifiers and the performance of models is compared based on their accuracies and F1 score. The model is trained using **TF-IDF Vectoriser**: TF-IDF indicates what the importance of the word is in order to understand the document or dataset.TF-IDF Vectoriser converts a collection of raw documents to a matrix of TF-IDF features. ngrams range is the range of number of words in a sequence.The simple TF-IDF model works well and gives importance to the uncommon words rather than treating all the words as equal in case of binary bag of words model.

Here the documents are also represented as vectors but instead of a vector of '0's and '1's, now the document contains scores for each of the words. These score are calculated by multiplying TF and IDF for specific words. So, the score of any word in any document can be represented as per the following equation: Therefore in this method, two matrices have to be calculated, one contain-

$$TFIDF(word, doc) = TF(word, doc) * IDF(word)$$

ing the inverse document frequency of a word in the whole corpus of documents and another containing the term frequency of each word in each document. The formulae to calculate both of them are as follows:

$$TF(word, doc) = \frac{Frequency of word \in the doc}{No. of words \in the doc}$$

$$IDF(word) = log_e\left(1 + \frac{No. of docs}{No. of docs with word}\right)$$

Under ngram model, vocabulary set like [”good” , ”car”, ”clean”, ”was”] will be considered as unigram.The ‘n’ in the n-gram approach represents a number, and it represents how many words there are in one gram.A bi-gram model will create a vocabulary set as [”car was”, ”was cleaned”, ”cleaned by”, ”by jack”, ... ,”by car”]. A bi-gram is a sequence of two words.The vector representation of the sentences will be [0 1 1 1 0 1] and [1 0 0 1 1 1] respectively.Now they are different, which is to say that the program now recognizes that those are two pretty different sentences because it now takes into account the sequences of words not just singular words. It doesn’t need to be limited to bigrams. It can be trigrams, four-grams, five-grams etc. we have used unigram bi-gram separately and also together to make better analysis of the text. We implemented tf-idf with unigram as well as bigram i.e. both the n-gram models separately and together.

In the unigram implementation only one word is taken at a time and tf-idf vector is constructed with this consideration. In the bigram implementation two words are taken at a time and then tf-idf vector is constructed, thus it also takes into account the sequence of words as they appaer in the dataset.

Next, we train the model using pre trained GloVe vectors and Word2Vec vectors. Word embeddings are words mapped to real number vectors such that it can capture the semantic meaning of words.They use some models to map a word into vectors such that similar words will be closer to each other. Pre trained word embeddings use the Neural Networks to train the word embeddings and get the word vectors. The word vectors are obtained by context, co-occurrence of the words , semantic and syntactic similarity.

The main idea of **GloVe** is to capture the meaning of a word embedding with overall structure of the entire corpus and to derive the relationship between them from global statistics. Glove generates word embedding by training a model based on global co-occurrence counts of words , global statistics and uses mean squared error as the loss function. The generated word embedding with such a model preserves word relationships and similarities. A co-occurrence matrix for a given sentence tells us how often a given pair of words appear together. Each element in the matrix is the count of the pair of the words occurring together.

GloVe can be used to find relations between words like synonyms, company-product relations, zip codes and cities, etc. It is also used by the SpaCy model to build semantic word embeddings/feature vectors while computing the top list words that match with distance measures such as Cosine Similarity and Euclidean distance approach. It was also used as the word representation framework for the online and offline systems designed to detect psychological distress in patient interviews.

The main theme of **Word2Vec** uses shallow neural networks to learn the embeddings. It is one of the popular word embeddings. that captures the context of words, while at the same time reducing the size of the data. Word2Vec is actually two different methods: Continuous Bag of Words (CBOW) and Skip-gram. In the CBOW method, the goal is to predict a word given the surrounding words.Here the model predicts the word under consideration given context words within specific window. The hidden layer has the number of dimensions in which the current word needs to be represented at the output layer. Skip gram is opposite of CBOW where it predicts embeddings for the surrounding context words in the specific window given a current word.The input layer contains the current word and the output layer contains the context words. The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer.

6 Results

We have used some of the most popular algorithms to train the classifier. The accuracy rate from each of the different algorithms for 50,000 features of data.

The below figures show the results that we got on training Naive Bayes classifier, Logistic regression classifier, SVM, XGB classifier and MLP classifier.

Naive Bayes trained on the feature extracted by the tf-idf vectoriser with ngram set to unigram. The Accuracy achieved by this trained model on the test set was 0.77.

Logistic Regression was trained on the feature extracted by the tfi-idf with ngram set to unigram. Accuracy achieved using this trained model on test set is 0.78. We get % accuracy using the Linear SVC and % using Logistic regression. As it is shown in the figure 1, logistic regression gives better accuracy and F1 score over Linear SVC model.

	precision	recall	f1-score	support
0	0.78	0.77	0.78	11980
1	0.78	0.78	0.78	11970
accuracy			0.78	23950

Figure 1: Logistic Regression using tf-idf unigram

From figure 1, it can be seen that using Logistic Regression using tf-idf ngram set to unigram achieved 0.78 accuracy and F1 score of 0.78.

As shown in Figure 2 shows comparison of the performance of the different models on the feature extracted by the tf-idf using different ngram set.

Naive Bayes trained on Tf- idf unigram, bigram and unigram and bigram together achieved the accuracy of 0.77, 0.73 and 0.78 respectively on the test set. Which shows that the Tf-idf feature vector extracted on the data increased the performance of the model and that training on separately using unigram model or bigram wasnt able to train on the data as better as training together.

Logistic Regression trained on Tf- idf unigram, bigram and unigram and bigram together achieved the accuracy of 0.78, 0.73 and 0.79 respectively on the test set. Which shows that the Tf-idf feature vector extracted on the data increased the performance of the model and that training on separately using unigram model or bigram wasnt able to train on the data as better as training together.

MODEL Tf-idf	ACCURACY	F1 SCORE
Naive Bayes (unigram)	0.77	0.77
Logistic Regression (unigram)	0.78	0.78
Naive Bayes (bigram)	0.73	0.71
Logistic Regression (bigram)	0.73	0.73
Naive Bayes (unigram and bigram)	0.78	0.77
Logistic Regression (unigram and bigram)	**0.79**	**0.79**
SVM (unigram and bigram)	**0.79**	**0.79**
XGBClassifier (unigram and bigram)	0.72	0.71
MLPClassifier (unigram and bigram)	0.76	0.76

Figure 2: Comparison of Model that uses Tf-idf

Also SVM model trained on Tf-idf unigram and bigram achieved the same accuracy as Logistic Regression of 0.79.

	precision	recall	f1-score	support
0	0.78	0.80	0.79	11980
1	0.80	0.78	0.79	11970
accuracy			0.79	23950

Figure 3: Logistic Regression using tfi-idf unigram and bigram

Above figure 3 shows the F1 score of 0.79 and accuracy of 0.79 on test set by the Logistic Regression model which was trained on the tf-idf model ngram set to unigram and bigram.

	precision	recall	f1-score	support
0	0.78	0.80	0.79	11980
1	0.79	0.78	0.78	11970
accuracy			0.79	23950

Figure 4: SVM tf-idf using unigram and bigram

Above figure 4 shows the F1 score of 0.79 and accuracy of 0.79 on test set by the SVM model

which was trained on the tf-idf model ngram set to unigram and bigram.

MODEL Word2vec	ACCURACY	F1 SCORE
Naive Bayes	0.52	0.51
Logistic Regression	0.52	0.31
SVM	0.50	0.51
XGBClassifier	0.59	0.59
MLPClassifier	0.50	0.60
CNN + Bi-LSTM	**0.76**	**0.76**

Figure 5: Model Trained on features extracted by Word2Vec

Above figure 5 shows the comparison of the models accuracy and f1 score. Naive Bayes, Logistic Regression, SVM, XGBClassifier, CNN+ bi-directional LSTM are trained on the features extracted by the Word2Vec. Naives Bayes accuracy achieved was 0.52 which is not as good as achieved in the model trained using Tf-idf. Also Logistic Regression model achieved only 0.52 again not as good as achieved when trained on Tf-idf features. SVM achieved 0.50 which performed worst among the models trained on the Word2Vec model. CNN+Bidirectional LSTM which is a deep learning model which achieved the accuracy of 0.76 when compared to the model trained on the Glove feature vector achieved a better performance.

Deep Learning model using GLOVE	ACCURACY	F1 SCORE
SimpleRNN	0.69	0.69
LSTM	0.73	0.72
CNN + Bi-LSTM	0.72	0.72

Figure 6: Deep Learning model on Glove feature vector

SimpleRNN, LSTM and CNN + Bidirectional LSTM model were trained on the Glove feature vector. SimpleRNN model achieved 0.69 accuracy, LSTM model achieved 0.73 accuracy while CNN + bi-directional LSTM model achieved 0.72 accuracy. when compared to the Deep Learning model trained using Word2Vec feature Vector, CNN+Bi-LSTM model performed better on Word2vec feature Vector.

7 Conclusion

In this report we followed the TF-IDF, Word2Vec and GloVe vector approach to classify tweets as either negative or positive after preprocessing. The pre trained vector approach is used so that we ourselves don't need to extract trainable features from the dataset. We can simply assign the values from these pre trained vectors and enter zeros for the words that are not present in these vectors.

As this is a classification task so we implemented Linear SVC and Logistic regression, Naive Bayes, MLP, XGB, and also some neural network models. The different n-gram implementation of tf-idf vector helped us analyse the differences of unigram, bigram and unigram + bigram approach. Unigram and bigram alone give reasonable accuracies but the unigram + bigram model gives best accuracy, i.e., 79%. The Word2Vec implementation gives better result for neural networks than the basic machine learning classifiers like Naive Bayes and Logistic Regression. It gives 76% accuracy in the CNN + bi-LSTM model. And the GloVe vector approach gives 73% as the maximum accuracy when fed to an LSTM recurrent neural network. It doesn't give better results than the Word2Vec vector.

Thus overall we reach a conclusion where we see that the unigram + bigram tf-idf implementation on Logistic Regression and SVM give the highest accuracy among all the models we trained. The accuracy attained by the unigram + bigram tf-idf implementation on Logistic Regression and SVM is 79% as indicated by the above table in figure 2.

8 Future work

Data Preprocessing using more parameters for better sentiments. Updating Dictionary for new Synonym and Antonyms of already existing words. Context Sentimental Analysis may be implemented in future for accuracy purposes. Also more pre trained vectors like BERT can be applied to see if better accuracy can be achieved. etter accuracy .We will try to clean the data in a way

that the model will be able to distinguish between the positive words and the corresponding negating words, eg. positive word - good, corresponding negating word - not good.

9 References

For code reference go to the urls:
https://drive.google.com/file/d/1l8jQDS9_H5bW_AxUTEaIbVoRsvDc0bIs/view
https://drive.google.com/file/d/1CUY5NBR0J3IjlLRI4srceeJpSXKn0XE5/view
For data reference go the url:
https://drive.google.com/file/d/1VOvwi-ewUH70fz5qywc94oETfAki1Y2l/view

References

[1] Bijoyan Das ,Sarit Chakraborty. *An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation.* Student Member, IEEE.

[2] Md. Rakibul Hasan ,Maisha Maliha, M. Arifuzzaman. *Sentiment Analysis with NLP on Twitter Data* Computer Communication Chemical Materials and Electronic Engineering (IC4ME2) 2019 International Conference on, pp. 1-4, 2019.

[3] Edilson A. Corrˆea Jr., Vanessa Queiroz Marinho, Leandro Borges dos Santos. *NILC-USP at SemEval-2017 Task 4: A Multi-view Ensemble for Twitter Sentiment Analysis* Institute of Mathematics and Computer Science University of S˜ao Paulo (USP) S˜ao Carlos, S˜ao Paulo, Brazil.

[4] Juan Ramos. *"Using TF-IDF to Determine Word Relevance in Document Queries"* pp. 1-4.

[5] Andrew L. Maas, Raymond E,Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. *"Learning word vectors for sentiment"* (2011).

[6] Michael Weigand, Alexandra Balahur, Benjamin Roth, Dietrich Klakow, Andres Montoyo. *"A Survey on Role of Negation in Sentiment Analysis"* pp 1-9.

.