In this project, I built a supervised learning model that can predict the polarity of a tweet. Details of the working procedure are as follows:

Dataset

The given dataset contains 1.6 Million tweets from Twitter. Each tweet has the following information- date, user, text, polarity (1 means “negative”, and 5 means “positive”). It has 0.8 million positive and 0.8 million negative tweets. As feature extraction part needs a lot time and computational resources to process this huge amount of data, I took a sample of 0.2 million tweets for this project. The selection of sample will be discussed in Exploratory Analysis part. Distribution of the tweets is as follows:

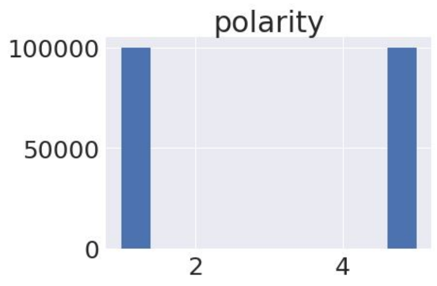

Figure 1: Tweet Distribution

Data Preprocessing

Each tweet has been preprocessed before fitting into the prediction model. A tweet may contains handler(user name), hashtag, url and emoticons. I replaced them using regular expression. For example, ‘https://t.com/abc’ has been replaced with ‘URL’ word, #hashtag has been replaced by the word(hashtag) itself. Moreover, any word with repeating characters (for example, happyyyyyyyy) has been replaced by the word without repeating character. Then a stemmer (Snowball Stemmer) has been applied to the tweets.

Feature Extraction

A variety of features can be used for tweet sentiment classification. Among them the most popular one is ngram. Using TFIDF vectorization, I extracted the ngrams of the tweets for n=1,2,3. As the dataset is pretty big, for avoiding the memory issue, I set minimum document frequency for 1-gram is 300, for 2-gram is 200 and for 3-gram is 50.

Moreover, using regex I tried to figure out whether the tweet contains any negative cues. It's important because negation cues can change the sentiment of the tweet.

Feature Importance

After feature extractions, I got around 850 features. All the features are not equally important. I ran Random Forest Classifier with default settings to find out the importance of the features. The most important feature I have found is W_1___handl (0.044654) which is the replaced word for any username. I plotted the top 30 feature importance scores against the features (Figure 2). Most of them are related to positive and negative type of words like sad, good, hate, sorry, bad, like, sick etc.

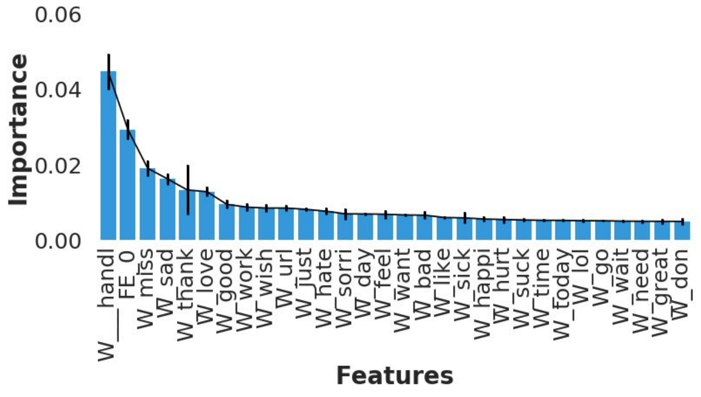

Figure 2: Feature Importance

Prediction Model

I used Naive Bayes as my supervised prediction model. There are many versions of the Naive Bayes algorithm. I implemented the Gaussian version of the Naive Bayes. Using python packages like numpy and math, I just built the function to calculate the likelihood, prior probability and posterior probability.

Validation

We split the whole dataset into training and testing sample. The split ratio is 33%. That means training data 67%(134000) and test data 33%(66000). After training the model with training data, I applied it on the test data and generated classification report. I got an accuracy of 73.80 % with 74% precision and 74% recall. I used all the features here. I also retried the process with 1-gram and negation features only. I got 72% precision and 71% recall then. The accuracy was 71.3% then. So adding 2-gram and 3-gram improved the overall performance.

Exploratory Analysis

As the dataset is pretty big, which needs much computational resources, I wanted to work with a smaller portion of data. But I also needed to look at the performance. That's why using the sklearn Support Vector Machine algorithm, I took different size of dataset and evaluated the performance (Figure 3). After that, I decided to work with dataset with 0.2 million size which ensures less drop of performance with a smaller portion of data.

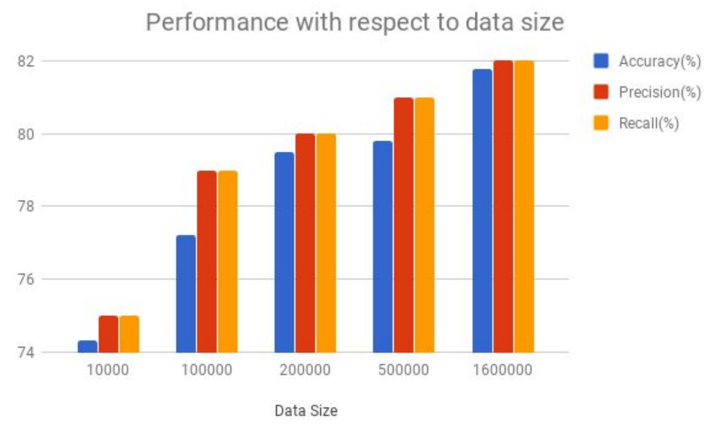

Figure 3: Dataset size vs Performance

Moreover, I compared my model with the SVM, Gaussian Naive Bayes and K-NN of sklearn python package. SVM performs better among all.

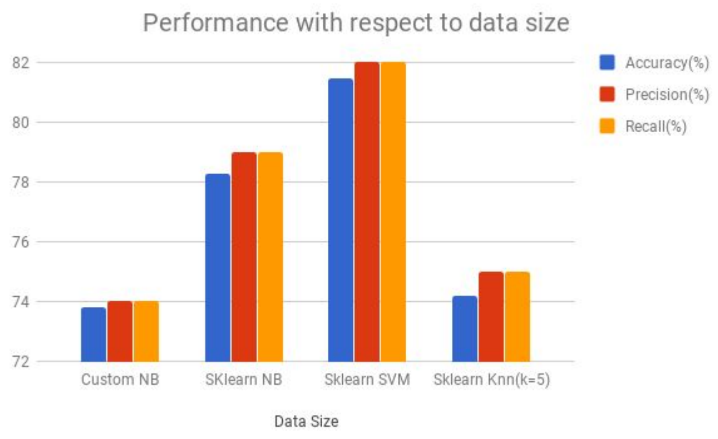

Figure 4: Comparison between different Algorithms