Machine Learning Engineer Nanodegree Lakshmy Menon
Capstone Project July 4, 2018

Sentiment Analysis on Tweets

Definition

Project Overview

Microblogging has become a very popular communication tool in today's world. The huge number of forums and availability of data makes it a perfect way to analyze the reaction of the end users as it happens. In this project the attempt is made to analyze the sentiments expressed as Tweets.

Problem Statement

The digital world is a great source for understanding the reaction of the readers around the world to an event. The goal of this project is to analyze and visualize the responses of the twitter users from reviews and comments in Twitter. The data available in twitter will be collected, cleaned, tokenized and classified as Positive, Negative and Neutral.

Following steps are involved in the project:

1. Download the tweets based on a Hashtag
2. Clean and tokenize the tweets for processing
3. Train the classifier using the tweets
4. Make a flask based UI for user to search and see the outcome

Metrics

Accuracy is a common metrics used for a classification. As it is simple and easy. As accuracy could be sometimes a little misleading due to the distribution in data we are using F1 score as well. As F1 score considers both precision and recall, it will give better measure of the performance.

Accuracy is simply the proportion of correctly classified instances. It is usually the first metric you look at when evaluating a classifier. The class labels in the training set can take on only 2 possible values, which we usually refer to as positive or negative. The positive and negative instances that a classifier predicts correctly are called true positives (TP) and true negatives (TN), respectively. Similarly, the incorrectly classified instances are called false positives (FP) and false negatives (FN).

Accuracy = (true positives + true negatives)/dataset size

Precision is a ratio of true positives (words classified as spam, and which are actually spam) to all positives (all words classified as spam, irrespective of whether that was the correct classification.

Precision = [True Positives/ (True Positives + False Positives)]

Recall is a ratio of true positives (words classified as spam, and which are actually spam) to all the words that were actually spam.

Recall = [True Positives/ (True Positives + False Negatives)]

F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.

F1 = 2 * (precision * recall) / (precision + recall)

Analysis

Data Exploration

An essential part of creating a Sentiment Analysis algorithm is to have a comprehensive dataset or corpus to learn from, as well as a test dataset to ensure that the accuracy of your algorithm.

The dataset consists of 5513 hand-classified tweets. These tweets were classified with 4 different topics.

Each entry contains:

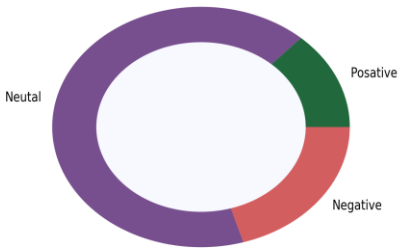
- Sentiment label: 'positive', 'neutral', 'negative', or 'irrelevant'
- Tweet text

The data that were classified as 'irrelevant' were removed and not processed further. Any tweets containing only irrelevant numbers or URL were removed.

Exploratory Visualization

The visualization of the available data and segments are showed below. This shows that we have more data with Neutral sentiment. The ratio of positive and Negative sentiments in both test and train data set remained the same.

The below given figure shows how the data is distributed in the Training dataset.



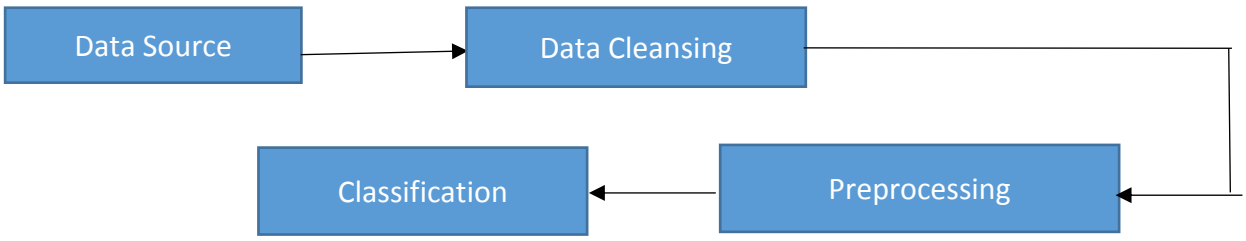
The below given figure shows how the data is distributed in the Testing dataset.

Algorithms and Techniques

Sentiment analysis is a common application of Natural Language Processing (NLP) methodologies, particularly classification, whose goal is to extract the emotional content in text. In this way, sentiment analysis can be seen as a method to quantify qualitative data with some sentiment score. Sentiment analysis can predict many different emotions attached to the text, but here only three major are considered: positive, negative and neutral.

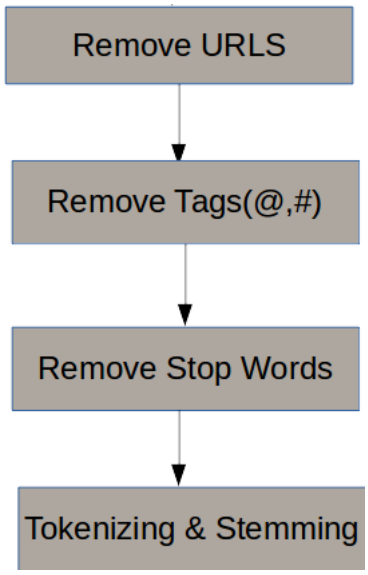
The first step is to collect the data from the data sources and then clean and preprocess the data as required. Once the data is ready the classification model will be applied and the accuracy and F1 score will be measured for different classification models.

The workflow for approaching a solution is given below.



Data Source
The Twitter data will be collected and used as the source of Data. The Sanders hand classified tweets will be used for training and testing the classifiers.

Data Cleansing and Preprocessing
The Cleansing of data is an important step as this enables better understanding of the sentiment. The below steps will show the cleaning and preprocessing steps that is used.



Word Embedding
Word embedding is the collective name for a set of language modelling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers.

The below two techniques are used here:

- ✓ Bag of Words
- ✓ Glove

Bag of Words
The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). Also known as the vector space model. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

The common method in this is N-gram model. Bag-of-word model is an order less document representation—only the counts of words mattered. For instance, in the above example "John likes to watch movies. Mary likes movies too", the bag-of-words representation will not reveal the fact

that a person's name is always followed by the verb "likes" in this text. Applying to the same example above, a bigram model will parse the text into following units and store the term frequency of each unit as before.

```
[
    "John likes",
    "likes to",
    "to watch",
    "watch movies",
    "Mary likes",
    "likes movies",
    "movies too",
]
```

Glove

GloVe, coined from Global Vectors, is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. It is developed as an open-source project at Stanford. Pre-trained vectors are available for download and use.

Classification

In this step different the classification Model will be applied. The different classifiers that are used in the project are given below.

1. Naive Bayes

The Naive Bayes classifier is the simplest and most commonly used classifier. Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes theorem with the “naive” assumption of independence between every pair of features. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.

The classifier more suited for text classification is MultinomialNB. This implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification.

This classifier is used in the project as the sample data size is small and this classifier is well suited for training on a small data size. The simple design of Naive Bayes classifiers make them very attractive. Moreover, they have been demonstrated to be fast, reliable and accurate in a number of applications of NLP.

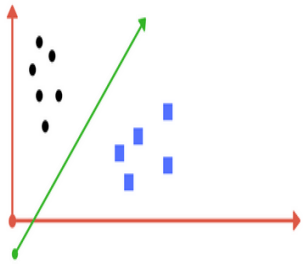
2. XGBoost

The XGBoost in supervised learning usually refers to the mathematical structure of how to make the prediction yi given xi. For example, a common model is a linear model, where the prediction is given by $\hat{y}_i = \sum_j \theta_j x_{ij}$, a linear combination of weighted input features. The prediction value can have different interpretations, depending on the task, i.e., regression or classification

The main advantage of using XGboost is that it's ease of use and its computational efficiency. The training and predicting is much faster than random forest and other such techniques. This technique is used in the project for the better scoring and faster prediction.

3. Support Vector Machine

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and detection. A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks.



The separation by hyper plane is shown above.
The SVM is widely used for language processing. It tends to have better generalization capability on unseen data than similar algorithms. The high dimensional structure with a sparse feature vectors works well for SVM. This classifier also works well with an imbalanced data.

Benchmark

There are a number of classifiers and methods applied on this dataset. The accuracies are given below for some of the classifications.

Naive Bayes
The accuracy of Unigrams is at 43.24% after applying crossvalidation.

XGBoost
The XGBoost seems to give an accuracy of 52.64%. This changes significantly with the crossvalidation.

Reference:
<< https://github.com/marrrcin/ml-twitter-sentiment-analysis/blob/develop/twitter_sentiment_analysis.ipynb>>

Methodology

Data Cleaning and Preprocessing

Data Cleaning

The first step was to load and clean the data. The URL's, numbers and mentions are removed so that the classifier could work on a significant set of words. The dataset for training and testing was downloaded as CSV files. In the UI, the tweets were stored as json file. The file then read and processed as needed.

Tokenizing and Stemming

Stemming is an attempt to reduce a word to its stem or root form. For example stem of the word working is work. There are number of algorithms available for stemming. The most commonly used ones are Porter stemmer and Snowball stemmer.
Tokenization is the task of chopping up a character sequence or a defined document unit into pieces, called tokens, perhaps at the same time throwing away certain characters, such as punctuation.

Removing Stop Words

Sometimes, some extremely common words (such as "the", "a", "an", "in") which would appear in high frequency and these words need to be excluded from the vocabulary entirely. These words are called stop words. We would not want these words taking up space in our dataset, or taking up valuable processing time.

Word Embedding

The words that were cleaned and processed was then converted to numerical vectors. The below methods were used for doing this.

Bag of Words

The most common method used in NLP to convert the words into a numeric representation is Bag of Words. The model learns a vocabulary from all of the documents, then each document by counting the number of times each word appears. This process is also called vectorization.

Different ways to convert text into vectors are:
1. **CountVectorizer**: This method works based on counting the number of times each word appears in a document. Then building a sparse matrix of the words.

2. **TF-IDF Vectorizer**: This method works based on calculating the frequency that each word appears in a document out of all the words in the document. The importance 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.

Both the techniques were used in the project, in order to create a better normalized vector for analysis.

GLOVE

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus. Thus we can convert word to vector using GloVe. The Stanford NLP Group has made their pre-trained GloVe vectors publicly available, and among them, there are GloVe vectors trained specifically with Tweets. There are four different versions of Tweet vectors each with different dimensions (25, 50, 100, and 200) trained on 2 billion Tweets.

The 200 dimensions pre-trained GloVe vectors was used in the project. The averaged word vectors were created using GLOVE and the result was passed to the classifiers.

Implementation

Data Source

The tweets to train and test the model was collected as CSV file and processed as text. The data collection for python based UI will be done using the Tweepy package available in python. The configuration and user keys that were created earlier was used to connect to Twitter and download the tweets related to the hashtag. The tweets were downloaded into the system and stored as a Json file. Once the download is complete the file will be opened and the tweets will be loaded and preprocessed.

Data Cleaning

In order to analyze and score the data we need to first clean and process the data. As the data we were using is from Twitter it may contain many characters that needs to be removed.
The Preprocessor library was used for cleansing the data. This library currently supports cleaning:

- URLs
- Hashtags
- Mentions
- Reserved words (RT, FAV)
- Emojis
- Smileys
- Numbers

Tokenizing and Stemming

Tokenizers divide strings into lists of substrings. The word_tokenize in NLTK Tokenizer Package was used for tokenizing the words.
Stemmers remove morphological affixes from words, leaving only the word stem. PorterStemmer module in NLTK Stem package was used for stemming. The module uses Porter stemmer algorithm for this.

Removing Stop Words

Stop words are plain and are filler words that are not helpful in the processing. We can remove them easily, by storing a list of words that you consider to be stop words. NLTK (Natural Language Toolkit) in python has a list of stop words stored in 16 different languages. This list was used to remove the stop words.

Bag of words

The bag-of-words model is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a features for training a classifier.

CountVectorizer

The CountVectorizer with unigram was used for bag of words classifier. The change into a bigram was not able to create any significant change in the accuracy. The *max_features* was changed from 200-800 and the value 250 was selected based on the results and execution time. The grid search with a CV as 10 was performed and the best parameters and values were selected based on this.

```
if IN_JUPYTER :
    from sklearn.model_selection import GridSearchCV
    parameters = {'vect__ngram_range': [(1, 1), (1, 2)],
                  'tfidf__use_idf': (True, False),
                  'clf__alpha': (1e-2, 1e-3),
                 }
    gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1,cv= 10 ,scoring='accuracy')
    gs_clf = gs_clf.fit(data_train, Sentiment_train)
```

```
if IN_JUPYTER :
   print gs_clf.best_score_
   print  gs_clf.best_params_
```

```
0.715971966064183
{'vect__ngram_range': (1, 2), 'tfidf__use_idf': False, 'clf__alpha': 0.01}
```

TF-IDF Vectorizer

TF-IDF stands for term frequency-inverse document frequency. It will transform a count matrix to a normalized tf or tf-idf representation. The only parameter that was changed from default was *use_idf*, which would enable inverse-document-frequency reweighting.

```
1.  class TwitterData_BagOfWords():
2.
3.      def __init__(self,tweets_train,tweets_test):
4.          self.tweets_train=tweets_train
5.          self.tweets_test = tweets_test
6.
7.      def Data_BagOfWords(self):
8.              print "Creating the bag of words...\n"
9.      # Initialize the "CountVectorizer" object
10.             vectorizer = CountVectorizer(ngram_range=(1,  2),analyzer = "word",   \
11.                                          min_df=.0025, max_df=.1, max_features=250 )
12.
13.             tweets_array=np.asarray(self.tweets_train)
14.             tweets_testarray=np.asarray(self.tweets_test)
15.             train_data_features = vectorizer.fit_transform(tweets_array)
16.             test_data_features = vectorizer.transform(tweets_testarray)
17.
18.     # Numpy arrays are easy to work with, so convert the result to an array
19.             train_data_features = train_data_features.toarray()
```

```
            test_data_features = test_data_features.toarray()
            tfidf_transformer = TfidfTransformer(use_idf=False)
            X_train_tfidf = tfidf_transformer.fit_transform(train_data_features)
            X_test_tfidf= tfidf_transformer.transform(test_data_features)
            terms = np.array(vectorizer.get_feature_names())
            return X_train_tfidf,X_test_tfidf,terms
```

Glove

The pre trained Glove was used for creating the vector. The average vectoring is done using 200 dimension vector.

```
class TwitterData_Vector(TwitterData):

    def __init__(self,Twitter_data):
        self.Twitter_data=Twitter_data

    def glove_vec(self):
        print "Reached in glove_vec "
        num_features=200
        # Preallocate a 2D numpy array, for speed
        #glove_twitter = api.load("glove-twitter-200")
        reviewFeatureVecs = np.zeros((len(self.Twitter_data),num_features),dtype="float32")
        glove_twitter = word2vec.KeyedVectors.load_word2vec_format(join(GLOVE_DIR, 'glove-twitter-200.txt'), binary=False)
        # Initialize a counter
        counter = 0
        for tweet in self.Twitter_data :
            reviewFeatureVecs[counter] = super(TwitterData_Vector, self).get_w2v_vec(tweet, num_features, glove_twitter)
            # Increment the counter
            counter = counter + 1

        return reviewFeatureVecs
```

Classifier

Creating a Training and Predicting Pipeline

To properly evaluate the performance of each model, it's important to create a training and predicting pipeline that allows a quick effective way to train models using various sizes of training data and perform predictions on the testing data.

The below given class was created for this purpose and used during the project. The *in_save* input parameter allows an option to save the model so that it can be used later for predictions.

```
class Def_Classifier(object):
    def __init__(self,X_train, y_train, X_test, y_test, in_classifier,in_save=False):

        self.X_train=X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.in_classifier = in_classifier
```

```
        self.in_save = in_save

    def class_pred(self):
        print "Training and Testing the classifier...\n"
        #list_of_labels = sorted(list(set(y_train)))
        list_of_labels = (self.y_train)
        model = in_classifier.fit(self.X_train, self.y_train)
        if self.in_save:
            f = open(filename, 'w')
            pickle.dump(model, f)
            f.close()
        predictions = model.predict(self.X_test)

        precision = precision_score(self.y_test, predictions, average=None, pos_label=None, labels=list_of_labels)
        recall = recall_score(self.y_test, predictions, average=None, pos_label=None, labels=list_of_labels)
        accuracy = accuracy_score(self.y_test, predictions)
        f1 = f1_score(self.y_test, predictions, average='weighted', labels=np.unique(self.y_test))
        return precision, recall, accuracy, f1_
```

Two different methods of vectoring was used for the experiments. The first experiment was using bag of words for vectoring whereas the second one was using GLOVE average vectoring to produce the final vectors.

Classifiers

The different classifiers can be easily modelled on data using the pipeline created above. The implementation of classifier Naive Bayes is given below.

```
in_classifier = MultinomialNB(alpha=0.01,fit_prior=False)
 def_Classifier= Def_Classifier(BagofWord_Train,Sentiment_train,BagofWord_Test,Sentiment_test,in_classifier)
 precision, recall, accuracy, f1 =def_Classifier.class_pred()
 df_class = df_class.append({'Classifier':'MultinomialNB','precision':precision,'recall':recall,'accuracy':accuracy,'f1':f1 }, ignore_index=True)
 print "================== Results ==================="
 print  "classifier:"
 print   in_classifier
 print  "======================================"
 print  "Precision:"+ str(precision)
 print
 print  "recall:"+ str(recall)
 print
 print  "accuracy:"+ str(accuracy)
 print
 print  "f1:"+ str(f1)
 print
```

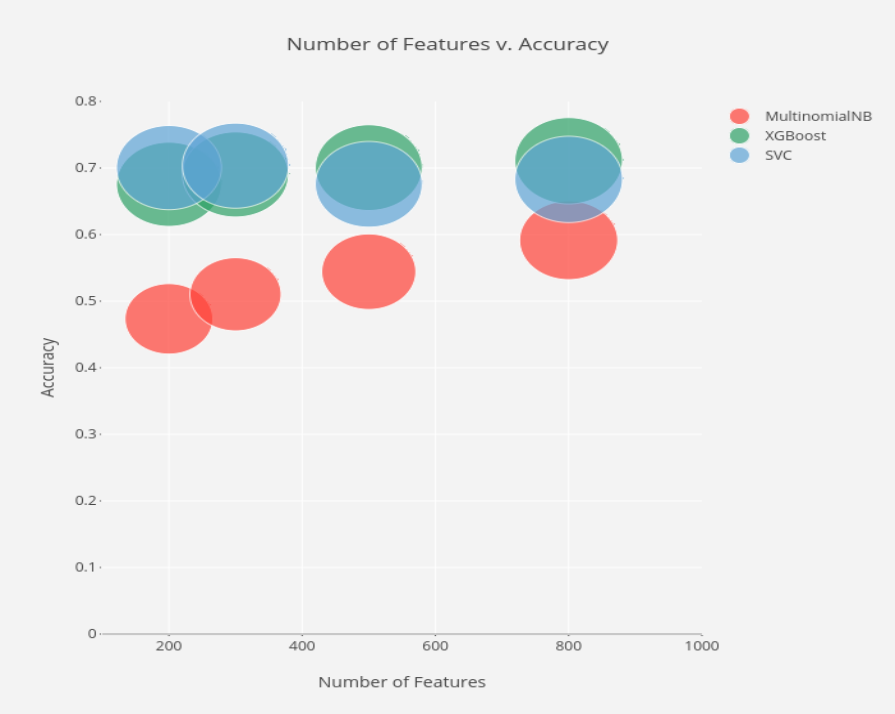
The MultinomialNB classifier was not used in Experiment 2 as the vectors contained negative values. All other Classifiers were used in both experiment 1 using Bag of Words and Experiment 2 using Glove.

Creating the UI using flask

A run.py file was created for executing the Twitter Search from flask. The Hashtag for searching once entered in the HTML page will be passed to the Run.py python file which will then process and create the final result using TwitterSearch class. The final result will be displayed as a graph in an HTML page.

Refinement

The experiment with Bag of words classifier was manipulated using multiple parameters in the CountVectorizer model. The accuracy and F1 score varied significantly as the parameter *max_features* was changed from 200 to 800. The different classifiers showed different results according to the value. The changes were noted and visualized as below. The other parameters did not show significant changes in the metrics. The Value 250 was selected based on the metrics and memory usage.



As per the visual we can see that the impact varied for the classifiers.

Naive Bayes

The NB classifier showed an increase in accuracy with increased number of features.

XGBoost

The XGBoost classifier showed an increase in accuracy with increased number of features.

Support Vector Machine

The SVM classifier showed a slight decrease in accuracy with increased number of features.

Results

Naive Bayes

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data.

Experiment 1-BOW

The main parameter that was optimized here was alpha and this was determined using the grid search. The alpha, smoothing parameter was given the value 0.1.
The result of the classifier is given below:

Classifier and parameter used	MultinomialNB(alpha=0.01, class_prior=None, fit_prior=False)
Precision	0.829
recall	0.438
accuracy	0.494
f1	0.518

Experiment 2-GLOVE

The MultinomialNB classifier was not used in Experiment 2 as the vectors contained negative values. MultinomialNB assumes that features have multinomial distribution which is a generalization of the binomial distribution. Neither binomial nor multinomial distributions can contain negative values.

XGBoost

XGBoost classifier parameters that were manipulated with significant changes in metrics were gamma, learning rate and objective. The final values were selected based on the metrics.
The result of the classifier is given below:

Experiment 1-BOW

Classifier and parameter used	XGBoostClassifier(n_estimators=440,max_depth=3,objective="reg:logistic",learning_rate=0.30,gamma=0)
Precision	0.709
recall	0.923
accuracy	0.681
f1	0.633

Experiment 2-GLOVE

Classifier and parameter used	XGBoostClassifier(n_estimators=600,max_depth=3,objective="reg:logistic",learning_rate= 0.3,gamma=0.0010)
Precision	0.744
recall	0.929
accuracy	0.722
f1	0.690

Support Vector Machine

SVM differs from the other classification algorithms in the way that it chooses the decision boundary that maximizes the distance from the nearest data points of all the classes.
The parameters that were manipulated with significant changes in metrics for this classifier were gamma, and kernel. After testing with both linear and RBF. The RBF kernel was chosen for better performance. The final values were selected based on the metrics.
The result of the classifier is given below:

Experiment 1-BOW

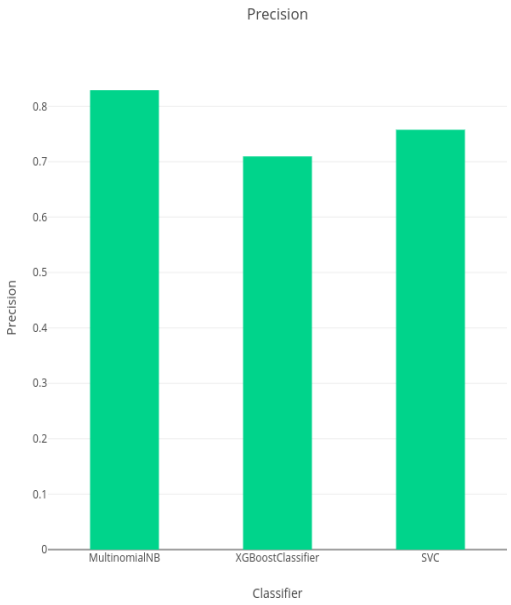
Classifier and parameter used	SVC(kernel='rbf', C=1E6,gamma= 0.001)
Precision	0.757
recall	0.898
accuracy	0.703
f1	0.678

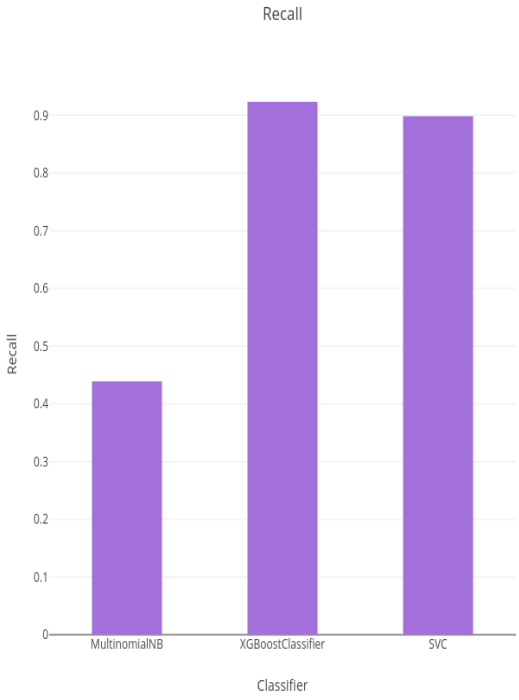
Experiment 2-GLOVE

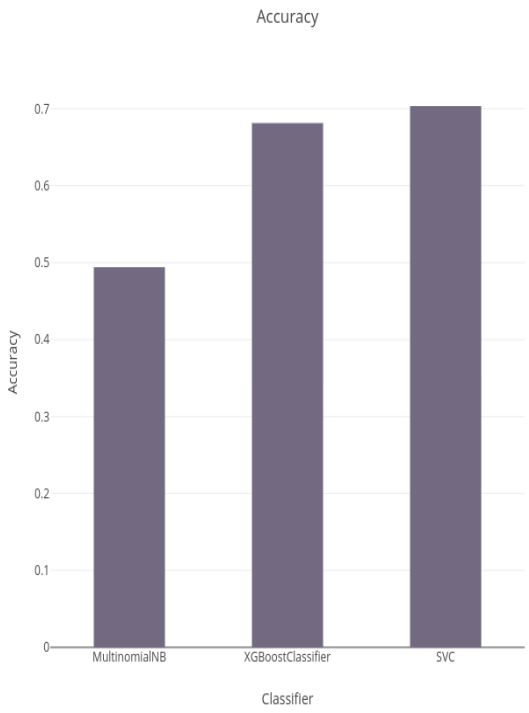
Classifier and parameter used	SVC(kernel='rbf', C=1E12,gamma=.001)
Precision	0.777
recall	0.871
Accuracy	0.719
f1	0.706

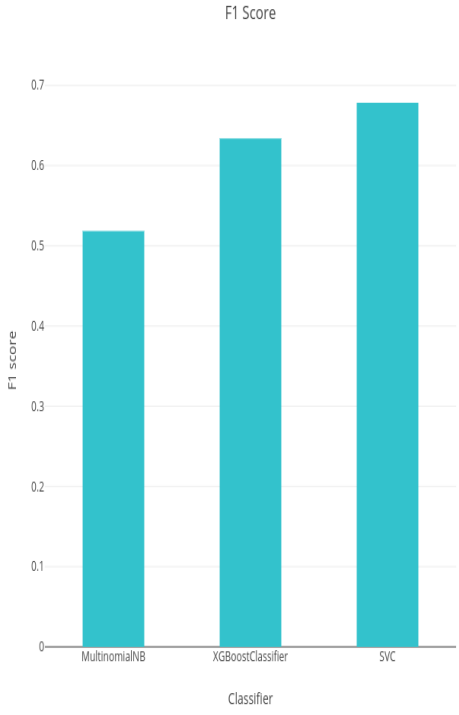
Visualizing the Metrics

Experiment 1-BOW

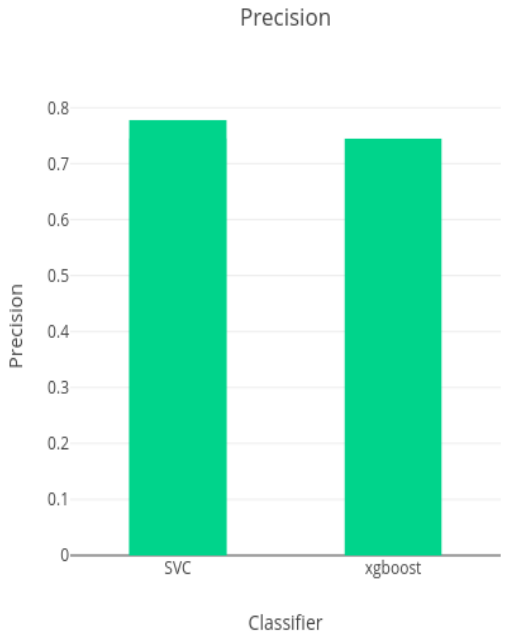


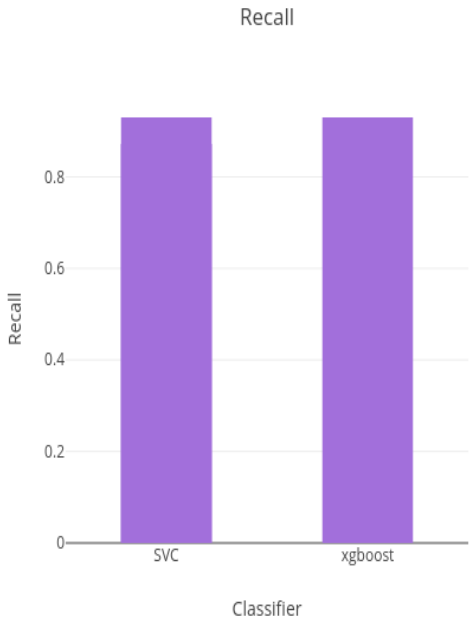


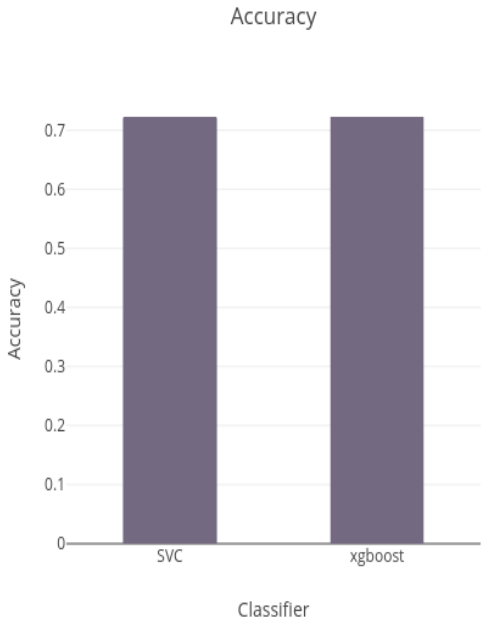


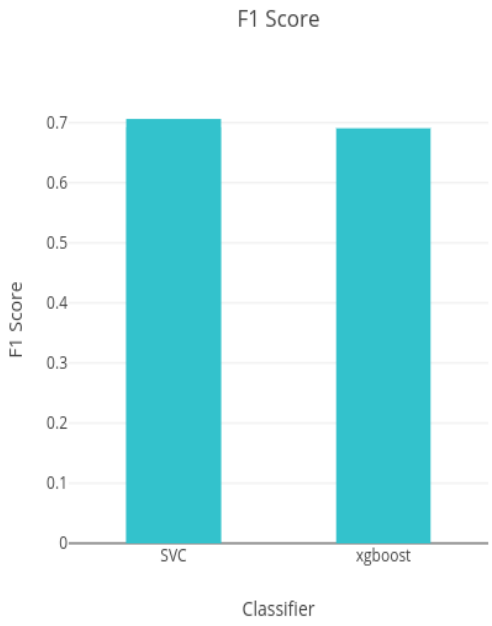


Experiment 2-GLOVE









The XGboost model with Glove was selected as the final model. This was chosen based on the F1 score, Accuracy and the execution time. The time required in order to download and predict was taken into account. The SVM model despite giving better metric score was more time consuming.

Justification

The final results vary from the benchmark results as some of the methodologies used here is different. In the case of Naive Bayes the classifier used in the benchmark is BernoulliNB and the one used in the project is MultinomialNB. The accuracy reported in the project is 49.4 which is lesser than the benchmark but it was also found that it can be improved to 59.14 by changing the parameters in CountVectorizer.

The XGBoost classifier is showing significantly better performance than the benchmark. The accuracy mentioned in benchmark was 52.64%.The project using glove instead of Word2vec and other feature extractions is showing an accuracy of 69%.

One of the reasons for not being able to provide much better result was the data set being too small. The preprocessing and cleaning the data reduced the data even further.

Conclusion

Reflection

The challenges were on deciding the perfect data source as there are a number of microblogging sites available. The twitter was selected due to the availability of the API that was widely used and the availability of information on the API. It was decided not to add other data sources as the averaging of vectors will become difficult with different text sizes and the models would be sensitive to the text size.

The number of researches and studies in Sentiment analysis was another challenge in clearly defining the problem and the scope of the project. There are a number of ways the project could have been defined and studied using multiple methods and embedding techniques. The final decision was made based on the system restrictions and availability of Data.

The final product is a good solution as it could be made into a small desktop application. This can be used by any person with basic knowledge in software. The final visualization is simple and easy to understand. The main advantage was to make sure that the application will run without high computing power and without consuming a lot of resources.

Improvement

Microblogging has become a very popular communication tool and millions of messages are appearing daily. The main services for microblogging are Twitter, Tumblr, and Facebook. In this project the analysis is performed on tweets alone. Other sources can be added to the project which will enable more data and better analysis and results. The API can be added as a new class such as the existing TwitterSearch class.

There are lot of researches done in the field of sentiment analysis. The Researches in sarcasm detection and emotion mining are some of the fields that could be used for improving the results.

The word embedding techniques like word2vec and other pre-trained models can be used for further studies. The neural networks and deep learning techniques can be explored as well. The deep learning methods are not used in this project mainly due to the Computing power and system limitation.

Reference:

Word embedding
<< https://en.wikipedia.org/wiki/Word_embedding>>
Benchmarking
<<http://zablo.net/blog/post/twitter-sentiment-analysis-python-scikit-word2vec-nltk-xgboost>>
Classification
<< https://stats.stackexchange.com/questions/169400/naive-bayes-questions-continus-data-negative-data-and-multinomialnb-in-scikit/220107>>
Bag of words
<< https://www.kaggle.com/c/word2vec-nlp-tutorial>>
<< https://en.wikipedia.org/wiki/Bag-of-words_model>>
Pipeline
<< http://scikit-learn.org/stable/auto_examples/model_selection/grid_search_text_feature_extraction.html>>
Glove
<< https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-10-neural-network-with-a6441269aa3c>>
NLP
<< http://ataspinar.com/2015/11/16/text-classification-and-sentiment-analysis/>>
Sentiment Analysis
<< https://ryan-cranfill.github.io/sentiment-pipeline-sklearn-2/>>
Jupyter notebook to application
<< https://danny.fyi/embedding-jupyter-notebooks-into-your-python-application-bfee5d50e4f8?gi=e6c866a93aa6>>