Twitter Sentiment Analysis

On

First GOP Debate Tweets and Airline Sentiment Data

“Data Analysis Project“

ECT-584 Summer II

By: Gunjan Pravinchandra Pandya (1547486)

➢ Introduction – Problem Statement

For ECT-584 Summer II class, I am going to do a Twitter Sentiment Analysis as a part of my final data analysis project and this falls under Web Content Mining branch of Wed Data Mining. Sentiment analysis is associated with identifying the associated sentiment of people associated with their tweets. Tweets, based on the words used, can be classified into positive, neutral or negative tweets and that is what the aim of this project is, to use the already classified training data to make a model and see how well it performs on the testing data to classify the tweets into pre-defined classes and this falls under umbrella of natural language processing (NLP). So this a supervised approach and I am going to use two datasets (more details below) which are from different domains but they reflect peoples positive, neutral or negative sentiments or reviews.

➢ Tools/packages

I am going to use Python for this data analysis project and the packages/libraries I have used are:

1) Pandas
2) Nympy
3) Natural language toolkit (Nltk)
4) Scikit-learn: Machine Learning in Python (Sklearn)
5) Matplotlib
6) String
7) Re
8) Time
9) Collections

➢ Datasets

Details of the two datasets I am going to use for this project are as follows:

1) First GOP debate sentiment analysis: Dataset with 13,872 tweets from https://www.crowdflower.com/data-for-everyone/ with following attributes: candidate, candidate: confidence, relevant_yn, relevant_yn: confidence, sentiment, sentiment: confidence, subject_matter, subject_matter: confidence, candidate_gold, name, relevant_yn_gold, retweet_count, sentiment_gold, subject_matter_gold, text, tweet_coord, tweet_created, tweet_id, tweet_location and user_timezone.
2) Airline Twitter sentiment: Dataset with 14,640 tweets from https://www.crowdflower.com/data-for-everyone/ with following attributes: _unit_id, _golden, _unit_state, _trusted_judgements, _last_judgement_at, airline_sentiment, airline_sentiment:confidence, negativereason, negativereason_confidence, airline,

airline_sentiment_gold, name, negativereason_gold, retweet_count, text, tweet_coord, tweet_created, tweet_id, tweet_location, user_timezone.

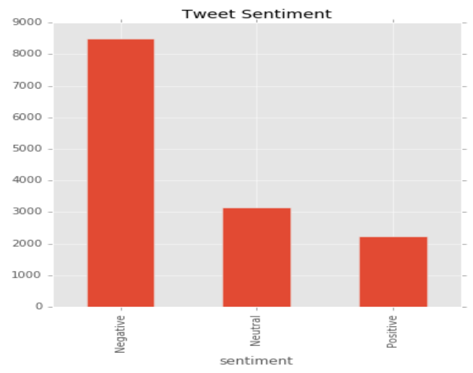
➢ Exploratory Data Analysis on the Datasets

So as a first step I will do some exploratory data analysis on the dataset to see how the distribution of the class is and to see what other insights we can derive regarding the data. Both of the datasets have already classified tweets into positive, negative and neutral classes. So I will do exploratory data analysis one by one on both the datasets.

1) First GOP debate sentiment analysis

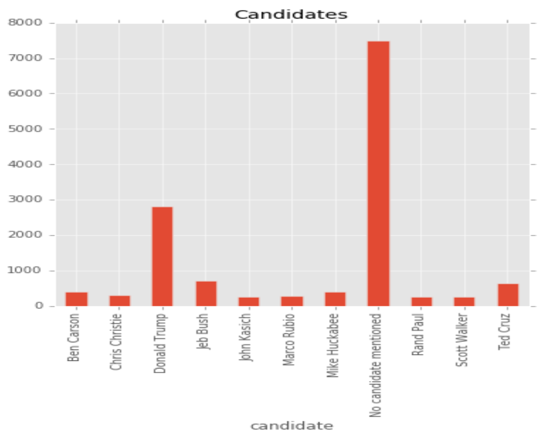
Number of tweets: 13,871

Class Distribution:



So we can see that in this data set the distribution is unequal and negative class dominates and the dominating sentiment in this data set is negative with around 8500 tweets and the count of neutral and positive tweet is around 3000 and 2000 respectively.

Candidate Distribution:

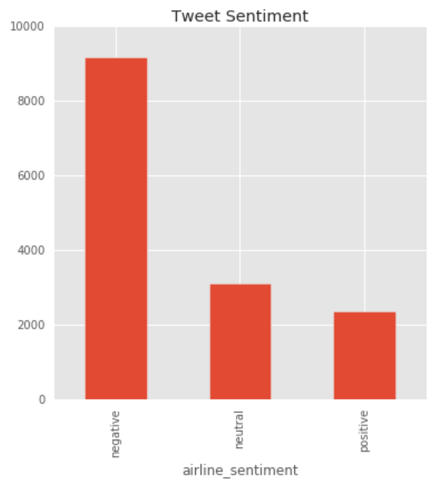


So we can see that in most of the tweets no candidate is mentioned in around 7500 tweets and if we talk about the tweets in which candidates are mentioned, highest number of tweets contain the mention of Donald Trump which aligns with what the trend that has been in this elections.

2) Airline sentiment data

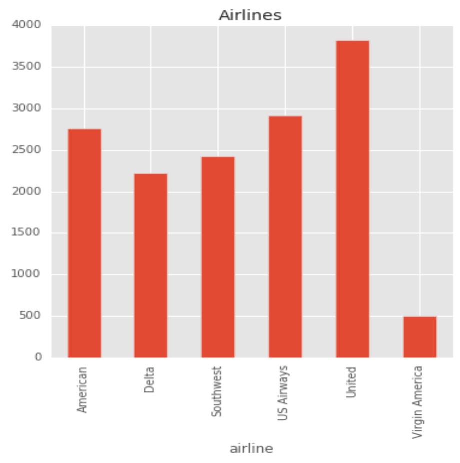
Number of tweets: 14,640

Class Distribution:



So we can see that in this data set the distribution is also unequal and negative class dominates and the dominating sentiment in this data set is negative with around 9000 tweets and the count of neutral and positive tweet is around 3000 and 2000 respectively.

Airline Distribution:



So we can see in the dataset that highest number of tweets are for United airlines and Virgin Atlantic is the least popular airline in terms of number of tweets.

So at an overall level in both the data sets we can see that the negative class is the dominating class and this will probably have an effect when we are creating models for our datasets as there will be more training data for negative class compared to positive and neutral classes and this might result into better accuracy in classifying negative classes on testing data compared to positive and neutral classes.

➢ Data pre-processing steps

Data pre-processing is the one of the most important part of this project as that is the step on which the accuracy of our final models will depend. NLTK library provides various options to preprocess text for natural language processing and I am going to use some of the features of NLTK like stop-words, lemmatization, punctuation etc.

Below I have outlined the pre-processing steps which I have performed on data and the purpose behind doing that:

1) Replacing hashtags (#) in tweets to spaces
➔ Hashtags are a part of tweets but they provide no help in the task of classification of tweets into positive, negative and neutral so I am replacing hashtags with white spaces.
2) Replacing all occurrences of "https:*"
➔ I am replacing all occurrences of any type of URL with format "https*" to a text "URL" and I am considering it as a stop word and removing it from the text of the tweets.
3) Replacing all occurrences extra and not useful words like "&amp, RT, URL" etc.
➔ I will compile a list of not useful words like &amp, RT and some of the punctuation marks which are not taken care by string.punctuation() and will replace all those by white space.
4) Converting all tweet text into lower case
➔ For uniformity in the text of the tweet I am converting all the text to lower case.
5) Lemmatization
➔ I am using WordNetLemmatizer function from NLTK to store the lemmatizations of the words in our tweets. Lemmatization is a process of combining synonyms or words which present the same meaning or show the same action.
6) Stop words removal
➔ The words which are most common in a language are stop words and so I would like to remove the stop words from tweets as they will help in no way during the classification process.

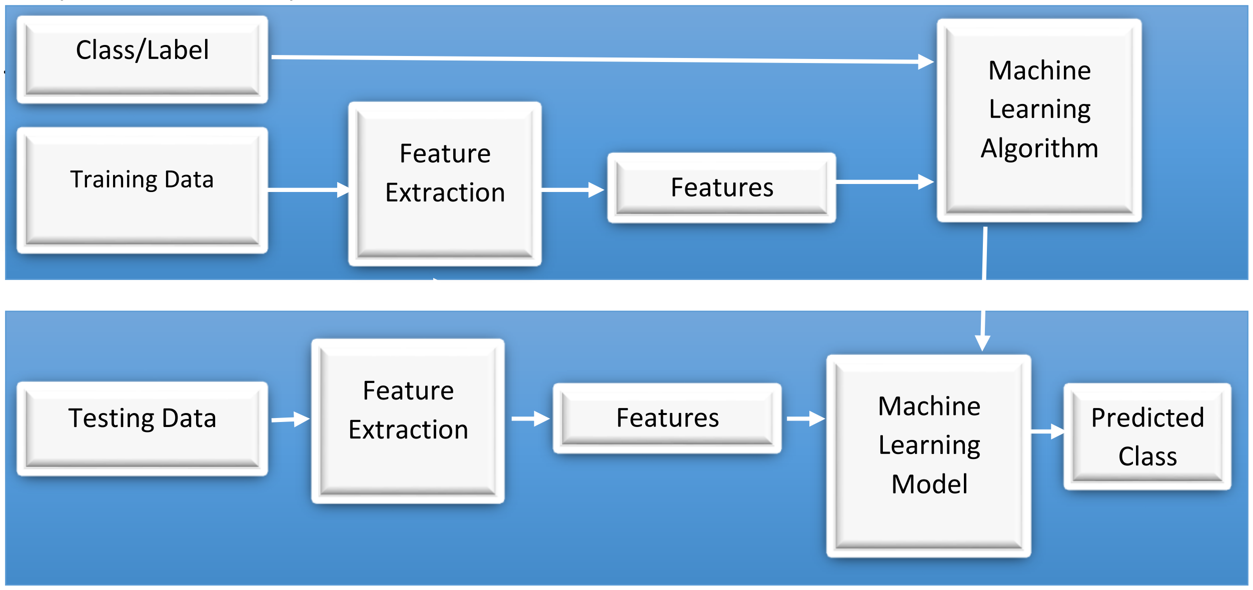
➢ Natural Language Processing - Feature Extraction

Here I am using TFxIDF Vectorizer from SKLEARN to do the feature extraction from text and it will create a matrix of term frequency and inverse document frequency features.

Also the other possible options to do feature extraction on text data using SKLEARN are by using unigram count vectorizer which considers words and their token counts and creates a term frequency matrix and the number of features it generates depends on the vocabulary size of the text, bigram count vectorizer considers a pair of two or more words depending on the parameters and uses them as feature to help in predicting the classes at a later stage and hashing vectorizer.

Reference for TFxIDF Vectorizer use: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

The process can be explained as below:



So after applying TFIDF transformation on the training dataset we can use the get_feature_names() method to see the extracted features and the corresponding TFIDF frequency and some of the features of dataset 1 include words like disappointing, donald, trump, win, hilary, influence, thanks, attack and many more and we can see that words like disappointing with high TFIDF frequency can definitely help in classifying tweets to negative classes as it is one the negative sentiments but words like trump, Donald, hilary and name of other candidates are the words which creates ambiguity and thus creates classification a difficult task as they can easily appear in negative, neutral or positive tweets.

Some of the features of dataset 2 includes words like delayed, good, cancelled, bad, response, thanks, stuck, wait, weather, worst and thus we can see that dataset 2 has better features and less ambiguity as words like delayed, bad, worst etc. clearly shows negative sentiments and words like good, response, thanks etc. can show positive sentiment but the point to think about here is that words when thought of being combined with word before or after that word might make a difference to sentiment like a tweet saying “not delayed” might indicate a neutral or positive sentiment but due to word delayed being occurring in many of the negative tweets can force a model to classify such a tweet into negative tweet and thus doing feature extraction using bigram counts can be very helpful in certain scenarios.

➢ Application of Algorithms

After doing feature extraction and creating TFxIDF matrix for the features in the tweets i.e. words in the tweets, now I am applying various classification algorithms to train the model and then apply it to testing data and check the accuracy. The algorithms I am going to use are:
1) Multinomial Naïve Bayes
2) Logistics Regression
3) k-Nearest Neighbors
4) Random Forest

I have split the data into training and testing and the split is an 80-20 split with 80% of data being the training data and 20% being the testing data.

Note: The accuracy value, counts in confusion matrix, precision and recall values might not reproduce when the code is executed at different machine. I have used random_state parameter wherever possible in an effort to be possible to reproduce the same results but values might change depending on the systems parameters or other factors.

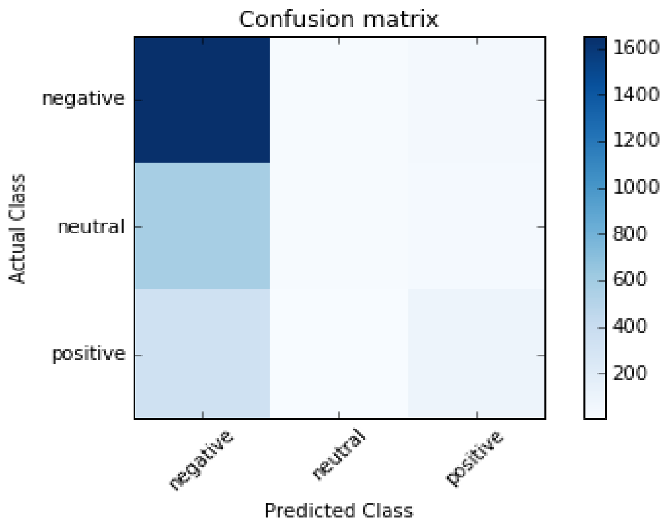
1) Multinomial Naïve Bayes

I have applied multinomial Naïve Bayes classification algorithm to create the model on the training data and have predicted the classes for the test data and the reported accuracy, confusion matrix, precision, recall and f1 score are from Python Output. This algorithm considers naïve assumptions of independence between the pairs of features.

1) Reported measures for tweets from debate data

Overall Accuracy: 63.78%

Confusion matrix:



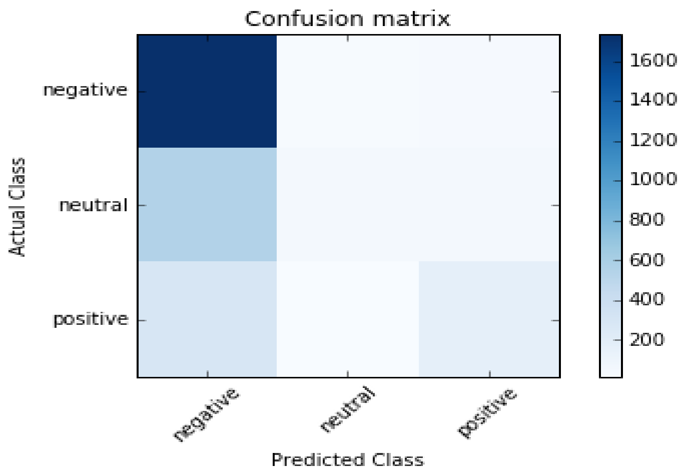
Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1663	17	39
	Neutral	583	13	24
	Positive	333	9	94

Overall Precision: 0.57
Overall Recall: 0.64
Overall F1 Score: 0.54

2) Reported measures for tweets from airline data

Overall accuracy: 68.85%

Confusion matrix:



Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1757	38	31
	Neutral	508	71	32
	Positive	291	12	188

Overall Precision: 0.68
Overall Recall: 0.69
Overall F1 Score: 0.63

So we can see from the reported accuracy measures that the performance of the Multinomial Naïve Bayes classifier is better on the airline data then the debate data and we received more accuracy, precision and recall on the airline sentiment dataset and overall time taken in training and predicting on testing dataset for both the dataset was 0.83 seconds. So we can see that the time required for training and prediction time i.e. querying the trained model is a very small and this is because of the naïve assumption of independence of features which this algorithm takes into consideration.

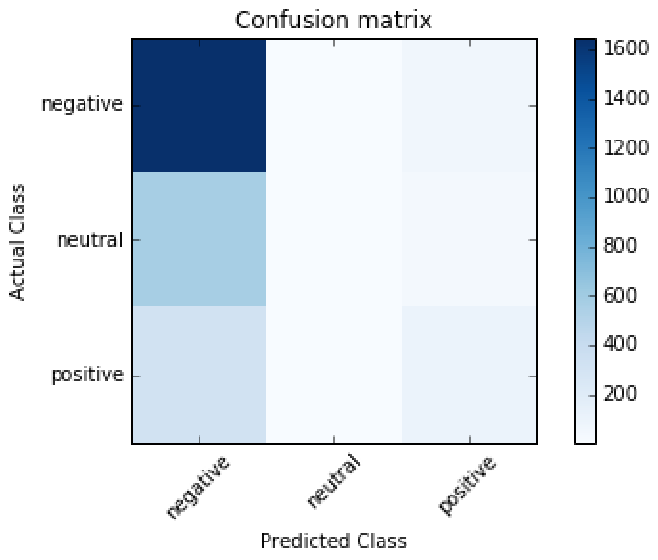
2) Logistics Regression

I have applied logistics regression algorithm to create the model on the training data and have predicted the classes for the test data and the reported accuracy, confusion matrix, precision, recall and f1 score are as follows (from Python Output). Multiclass option can be set to multinomial and ovr (One-vs-rest), when ovr for each label a binary problem is fit, else multinomial loss is fit across the probability distribution. Here I have selected Multinomial option.

1) Reported measures for tweets from debate data

Overall Accuracy: 64.40%

Confusion matrix:



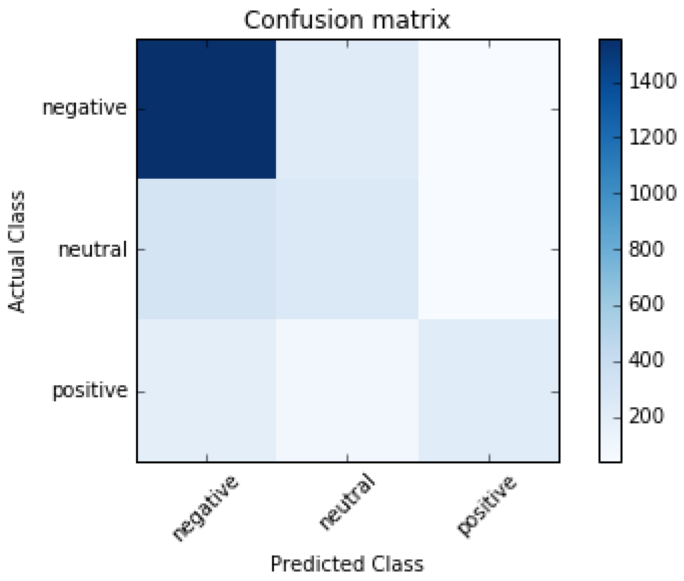
Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1648	12	59
	Neutral	558	25	37
	Positive	316	6	114

Overall Precision: 0.62
Overall Recall: 0.64
Overall F1 Score: 0.55

2) Reported measures for tweets from airline data

Overall accuracy: 69.16%

Confusion matrix:



Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1555	223	48
	Neutral	314	253	44
	Positive	188	86	217

Overall Precision: 0.68
Overall Recall: 0.69
Overall F1 Score: 0.68

So we can see from the reported accuracy measures that the performance of the Multinomial Logistics Regression is better on the airline data then the debate data and we received more accuracy and precision and recall on the airline sentiment dataset and overall time taken in training and predicting on testing dataset for both the dataset was 2.57 seconds. So if we compare it with Multinomial Naïve Bayes it takes more time to train and test the model. Also the performance on the neutral class in debate data is very poor by this model which was the same by Multinomial Naïve Bayes but if we compare the performance on airline data set logistics regression was able to predict neutral classes with better accuracy. The poor performance on the neutral set is a result of less training data as well as ambiguous word set in the neutral tweets.

3) K-Nearest Neighbors

I have applied k-nearest neighbor classifier to create the model on the training data and have predicted the classes for the test data and the reported accuracy, confusion matrix, precision, recall and f1 score are as follows (from Python Output). The distance measure I have used is "minkowski" as it gives the best results compared to “euclidean” and “manhattan” for the real-valued vector space we have.

Also I have used various combinations of k’s starting from 5 to 14 and the observed training and testing accuracies are as follows for both the datasets:

k-Nearest Neighbors	**Accuracy (%)**			
	Datatset1 (Debate Data)		**Dataset2 (Airline data)**	
	Training	**Testing**	**Training**	**Testing**
5	64.99	60.43	71.00	67.49
6	65.13	60.90	70.55	68.03
7	65.01	60.76	70.30	67.59
8	65.11	61.62	70.41	67.32
9	65.10	62.16	70.08	67.18
10	64.96	61.84	70.18	67.25
11	64.66	62.05	70.22	67.01
12	64.70	62.81	69.85	67.28
13	64.67	62.56	69.55	66.97
14	64.61	62.63	69.46	67.18

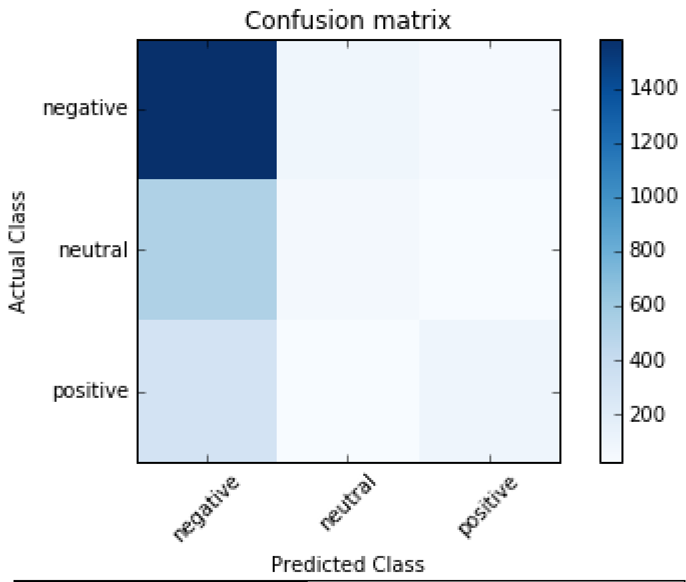
The ideal value of k in my opinion is k = 12 for dataset 1 as increasing value of k after 12 results in decrease in training accuracy as well as testing accuracy falls down which indicates we might overfit the model if we further increase the number of neighbors. So to avoid overfitting and to keep the computational complexity less (compared to higher values of k) it is in best interest not to increase value of k. Also the training and testing accuracy are within 10% which is desired.

The ideal value of k in my opinion is k = 7 for dataset 2 as increasing value of k after 7 results in decrease in training accuracy as well as testing accuracy falls down which indicates we might overfit the model if we further increase the number of neighbors. So to avoid overfitting and to keep the computational complexity less (compared to higher values of k) it is in best interest not to increase value of k. Also the training and testing accuracy are within 10% which is desired.

1) Reported measures for tweets from debate data

Overall Accuracy: 62.81%

Confusion matrix:



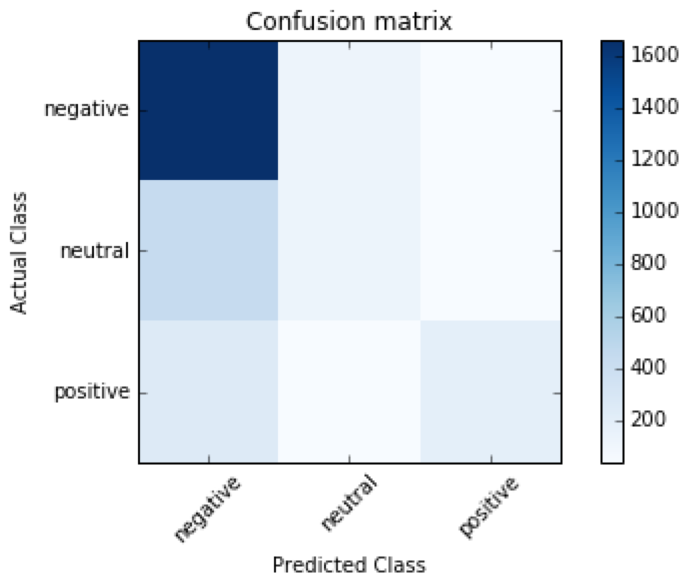
Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1583	87	49
	Neutral	531	61	28
	Positive	308	29	99

Overall Precision: 0.57
Overall Recall: 0.63
Overall F1 Score: 0.56

2) Reported measures for tweets from airline data

Overall accuracy: 67.59%

Confusion matrix:



Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1660	119	47
	Neutral	442	127	42
	Positive	251	48	192

Overall Precision: 0.64
Overall Recall: 0.68
Overall F1 Score: 0.64

So we can see from the reported accuracy measures that the performance of the k nearest neighbor is better on the airline data then the debate data and we received more accuracy, precision and recall on the airline sentiment dataset and overall time taken in training and predicting on testing dataset for both the dataset was 10.79 seconds. So if we compare it with Multinomial Naïve Bayes and Logistics regression it takes more time to train and test the model and this is the expected behavior because in k-nearest neighbor the distance or similarity to the nearest neighbors are calculated and then the testing sample is assigned to the corresponding class based on the classes of its neighbors and thus the run time for this algorithm more than 10 times the run time for other algorithms which we analyzed earlier. Also the k value will affect the time required for computation and larger values of k will result into higher computational times.

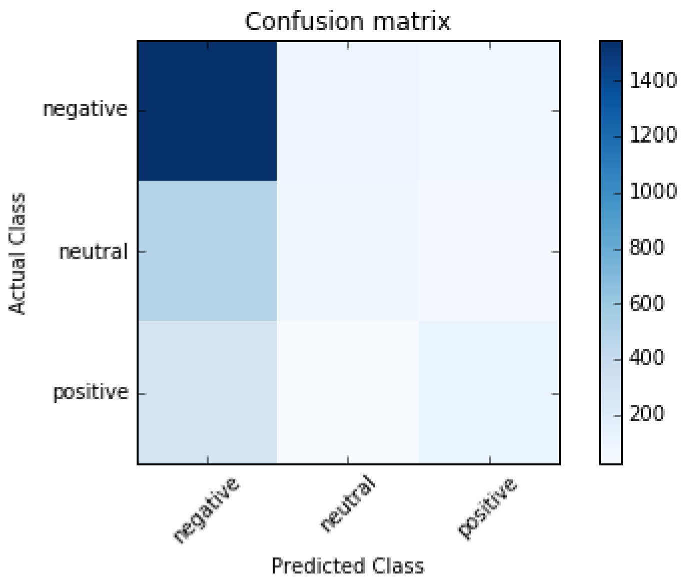
4) Random Forest

I have applied random forest classification algorithm to create the model on the training data and have predicted the classes for the test data and the reported accuracy, confusion matrix, precision, recall and f1 score are as follows (from Python Output). Random forest classifier fits multiple decision trees on various sample of dataset derived from the original dataset and uses averaging of the results of these trees, so it can be called a tree with multiple decision trees as its branches.

1) Reported measures for tweets from debate data

Overall Accuracy: 63.17%

Confusion matrix: (Using http://scikitlearn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html)



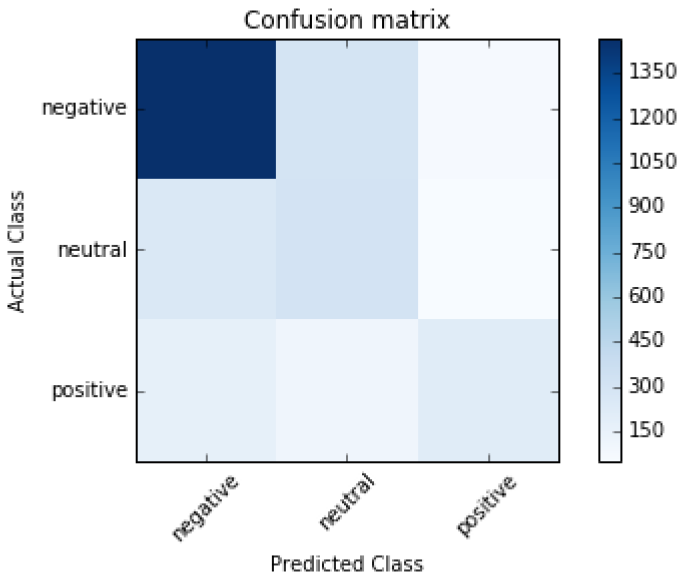
Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1548	97	74
	Neutral	495	86	39
	Positive	290	27	119

Overall Precision: 0.58
Overall Recall: 0.63
Overall F1 Score: 0.58

2) Reported measures for tweets from airline data

Overall accuracy: 67.83%

Confusion matrix:



Confusion Matrix		Predicted Class		
		Negative	Neutral	Positive
Actual Class	Negative	1466	296	64
	Neutral	255	306	50
	Positive	167	110	214

Overall Precision: 0.68
Overall Recall: 0.68
Overall F1 Score: 0.68

So we can see from the reported accuracy measures that the performance of the Random Forest algorithm is better on the airline data then the debate data and we received more accuracy and precision and recall on the airline sentiment dataset and overall time taken in training and predicting on testing dataset for both the dataset was around 6 seconds. So if we compare it with Multinomial Naïve Bayes and Logistics regression it takes more time to train and test the model and this is the expected behavior of Random Forest as it takes time to create a number of decision trees while learning the classification in data or creating the model.

➢ Comparison of Algorithms Used

So in the table below I have consolidated the accuracy measures and runtime for each of the algorithms we have applied on the two datasets.

Algorithm	Dataset 1			Dataset 2			Runtime (seconds)
	Accuracy (%)	Precision	Recall	Accuracy (%)	Precision	Recall	
Naïve Bayes	63.78	0.57	0.64	68.85	0.68	0.69	0.83
Logistics Regression	64.40	0.62	0.64	69.16	0.68	0.69	2.57
k-Nearest Neighbors	62.81	0.57	0.63	67.59	0.64	0.68	9.96
Random Forest	63.17	0.58	0.63	67.83	0.68	0.68	5.73

So here we can see that the accuracy of all the algorithms on the testing dataset is almost same and there is very minor or almost no difference in reported accuracies for dataset 1 and dataset 2 where accuracies range from 62.81% to 63.78% for dataset 1 and 67.59% to 69.16% for dataset 2. Logistics Regression gives best results on both dataset 1 and dataset 2.

If we talk about the precision, which is the fraction of the events where we correctly classified a tweet into one class out of all the classified tweets into that class, reported for all the algorithms we can see that the precision is same for all three algorithms Naïve Bayes, Logistics Regression and Random Forest which is around 0.57-0.58 while it is 0.62 for Logistics Regression on dataset 1. For dataset 2, the highest precision we get is 0.68 which is for Logistics Regression, Naïve Bayes and Random Forest while it is 0.64 when we used k-Nearest neighbors. If we compare Recall values, which is fraction of the events where we correctly classified tweets into a class out of all the tweets in that class, it is 0.64 for Naïve Bayes and Logistics Regression while highest recall we get for dataset 2 is 0.69 using Logistics Regression and Naïve Bayes algorithms. So the precision value here will specify how right we are predicting at the overall level and is the measure of exactness, so precision of 0.62 for Logistics Regression for dataset 1 (debate data) shows that for example if we predict that 100 tweets are negative, algorithm was right 62 of the times and 38 times the tweet didn't belonged to class it was assigned and belongs to positive or neutral class and recall shows how well we are doing in a particular class and is the measure of completeness, so recall of 0.69 shows that for example out of 100 tweets in negative class we were able to correctly classify 69 tweets. Here the precision and recall values are overall values which is an average over all three classes and individual values are visible in Python output when we run the code. One particular thing to notice when doing this project was that I was getting high recall for negative class and very low recall for neutral class which says that we were able to predict/classify almost all the tweets into negative class which actually belonged to negative class but all the neutral tweets were getting classified as either positive or negative tweets, mostly negative. The reason behind that is the presence of the ambiguous and not so positive or negative terms in the neutral tweets and large training data available for negative tweets which created a bias towards the negative class and thus the neutral tweets were getting classified as negative tweets. Also same is the case with positive tweets but the recall we have is better compared to neutral tweets which shows the presence of strong positive words in tweets but less availability of training data is what creates issues for the algorithms.

Also if we take a look at the confusion matrix created above for dataset 2, we can see that Naïve Bayes and k-Nearest Neighbor were very successful in predicting negative tweets into right class but their performance was very poor on other two classes while Logistics Regression and Random Forest were able to predict neutral and positive classes better compared to Naïve Bayes and k-Nearest Neighbor.

Also if we compare the runtime of all the algorithms we can see that the most computationally expensive algorithm is k-nearest neighbor as it needs to find all the nearest neighbors and then classify the tweet for each tweet in the dataset, so that takes more time comparatively to all other algorithms whereas Naïve Bayes is the fastest algorithms in terms of training and prediction times as it takes naïve assumptions into consideration that the events or features are not dependent and computes the probabilities of each tweet being classified into positive, neutral or negative class.

➢ What Else Can Be Done?

Natural language processing is a field in which continuous research and technological advancement takes place and so there are various possible methods using which we can improve the model accuracy and which can result into accurate classifications of tweet sentiments. Using Unigram counts, Bigram counts and Hashing vectorizer are the few alternatives to TFxIDF vectorizer which I have used for feature extraction from text data. Also using other fields present in the dataset like confidence of tweet to weight the counts or TFxIDF or unigram/bigram counts, might result into improvement in the accuracy. Also in addition to feature extraction, creating pre-defined set or bag of words and putting normally used words corresponding to positive and negative sentiment from dictionary into those sets and using them and the tweet data to classify tweets might also help in improving model accuracy. Also classifying tweets into objective and subjective is an interesting task and further help in sentiment analysis and text mining. One further application can be classifying tweets into angry, sad and happy to know a different set of sentiments from the tweets and there are numerous applications of this branch of web data mining into different domains.

➢ References

NLTK: http://www.nltk.org/
TFIDF Vectorizer: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
Confusion Matrix: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
SKLEARN: http://scikit-learn.org/stable/