Sentiment Classification of Tweets

Sentiment classification of written text is a challenging problem for machines to master. Accurately identifying whether the author of the text is implying a positive or negative sentiment is even more difficult without the text conveying a lot of emotions. Additionally, some aspects of human emotions like humor and sarcasm do not translate well into written text. Classifying tweets is even more challenging because of the limited number of words that can be represented in 140 characters. Also, the problem gets even more challenging for a machine because of the abundance of emoticons, spelling errors, short forms, internet slangs, interspersed numbers instead of grammatically correct sentences.

This classification problem is interesting because of the challenges it faces and also the abundance of applications if done correctly. The problem is akin to computers' understanding of human communication. Even though human emotions are not always binary or expressed in terms of negative or positive sentiment it is the basic building block of human emotion. If machines are able to decipher sentiment it could make our interaction with them much smoother. Additionally, we can leverage the power of computers to analyze text in a large scale.

We can use the technology to figure out if some communities or groups are having issues either it be geopolitical, or natural calamities and respond quicker. We could also analyze opinions in a large scale through the social media such we can figure if certain public policies are popular or unpopular. People tend to post unfiltered data on the web. We could use that to glance at the social media landscape to find which cities, states or countries are happier, economically prosperous. Instead of conducting polls which could be biased because people may not be forthcoming when they think they are being recorded. Sentiment analysis also could be used to find which products in the market popular or unpopular providing companies with perspective on what to expect.

In this regard, even though the problem is challenging to solve it has a lot of rewards. The data used for the classification problem consists of tweets that have been hand-labelled either positive or negative based on the sentiment conveyed by them.

The downloaded data contained random samples of tweets from a lot of users and has been categorized into positive and negative based on the sentiment expressed by the tweet. So, it could be biased by the emotions of the person categorizing the tweets. In order to process the data he data has been stripped of any emoticons and only contains alpha-numeric characters.

Examples of positive tweets:

- Juuuuuuuuuuuuuuuussssst Chillin!!
- thanks to all the haters up in my face all day! 112-102
- Feeling strangely fine. Now I'm gonna go listen to some Semisonic to celebrate
- (: !!!!!! - so i wrote something last week. and i got a call from someone in the new york office... http://tumblr.com/xcn21w6o7

Examples of negative tweets:

- i think mi bf is cheating on me!!! T_T
- .. Omgaga. Im sooo im gunna CRy. I've been at this dentist since 11.. I was suposed 2 just get a crown put on (30mins)...

- I missed the New Moon trailer...
- this weekend has sucked so far

Out of the almost 12K data points there are roughly 4500 tweets that have been labelled negative and the rest of 5428 tweets have been labelled positive. I have divided the dataset into 8K for training and 2K for testing and also kept another 2K as a holdout set for 1 fold validation. Tweets have been scrubbed for numbers and other punctuations so that only space separated words remain in the tweets. For example the following tweet:

> ".. Omgaga. Im sooo im gunna CRy. I've been at this dentist since 11.. I was suposed 2 just get a crown put on (30mins)..."

Is scrubbed by removing numbers and punctuations as such:

> "Omgaga Im sooo im gunna CRy Ive been at this dentist since I was suposed just get a crown put on mins"

The data is broken down into word vectors using a normal stemmer and getting rid of English stop words. Word count alone is not a good measure to classify positive and negative sentiments because common words of English show up on both the positive and negative tweets. Instead, a vector of its term frequency against its inverse document frequency (TF-IDF) is created such that unique words in both the negative and positive tweets have more weights. The vector consists of sparse data with mostly 0's. Once the data has been dummy coded for classification 0 for Negative sentiment and 1 for positive sentiment it is then fed into various learning algorithms.

Dataset contains 12,000 tweets which are divided into sixty percent training set and twenty percent of it into testing set and a holdout set similar to training set which is used for 1-fold cross validation.

Data Set:

	Rows	Features
Training	8000	15065
Testing	2000	15065
HoldOut	2000	15065

Decision Tree

The dataset was then fed into a decision tree to analyze if the tweets could be divided into positive and negative sentiment. DecisionTreeClassifier from Scikit-learn library was used without specifying the 'max_depth' parameter such that the tree expands until all the leaves are pure. It also uses Gini impurity by default. The classifier gave an accuracy of 64% with the depth of 1500 nodes.

Since the tree was too big and did not give impressive results the tree was iteratively pruned. The tree was pruned and the accuracy was calculated against the hold out set. The pruning accuracy has been plotted in the graph below. It shows that decision tree of depth 450 gave better accuracy on the hold out set to the tree with 1500 node depth.

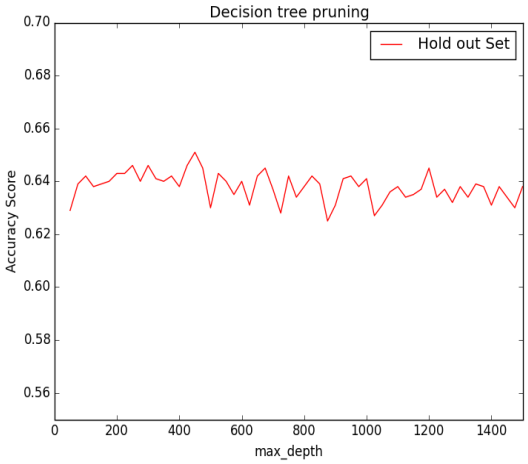


Fig 1: Accuracy of tree after pruning

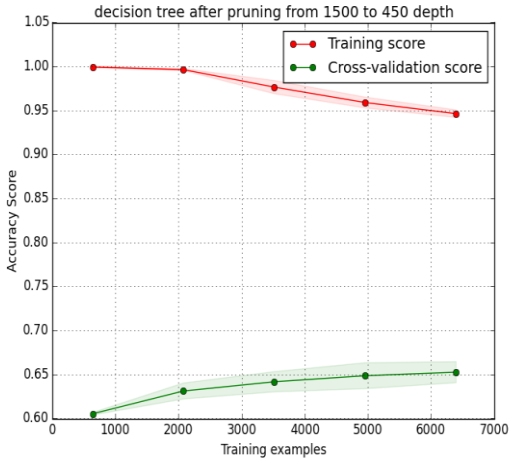


Fig 2: Learning curve for 5 set cross Validation

Since, pruning from 1500 node depth to 450 node depth increased accuracy from 64% to 65% that was used to build the final classifier. A 5 way cross validation was also done with the training set and the graph which is included above. Where we can see that as the size of the data is increased accuracy of training set decreases and accuracy of testing set increases. The classifier with 450 max_depth was built and tested on the test set of 2000 rows. The following table shows various metrics for the classifier:

	Training	Testing
Accuracy	0.938	0.646
F1 Score	0.945	0.681
Avg. Precision	0.996	0.712
Avg. Recall	0.949	0.758
Roc Auc score	0.932	0.642

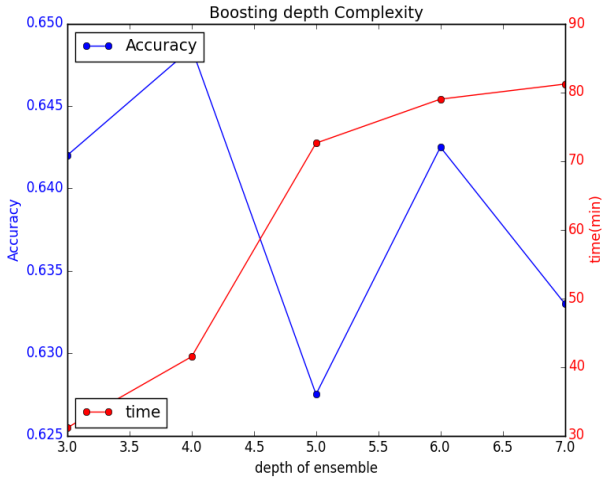
Pruning the tree did not change the accuracy by a lot because most important attributes are used near the root of the tree. Even though this has a high number of features there are a limited number of words that are positive or negative. These attributes are closer to the root as such pruning does not improve the accuracy by a lot

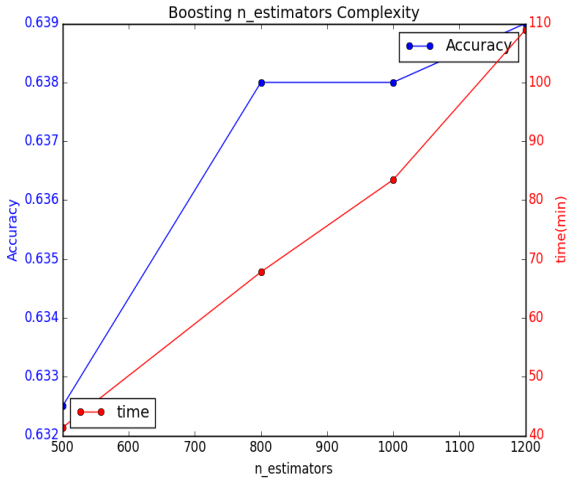
Boosting

Boosting was implemented through the AdaBoost classifier in the Scikit-learn library. In order to model the complexity various classifiers were trained by varying the number of estimators and also by varying the depth of the base estimators. The answers have been plotted below to show the effect of changing depth and number of estimators. Depth was kept constant while varying the number of estimators and vice versa.

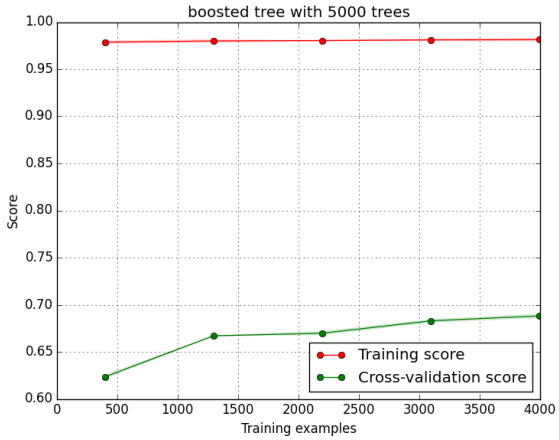
It can be inferred from the curves that changing the number of estimator increases the time needed to train the algorithm. Additionally, the accuracy of the classifier also increases. However, changing the

depth of the base estimator does not always increase the accuracy on the testing set. It could be because the base estimator is not optimal and creating





After the initial investigation various numbers of combination were tried amongst them base estimator with a depth of 4 and 5000 estimators gave the best results in a time of 7 hours. It gave an accuracy of 68% which is better than other classifiers.

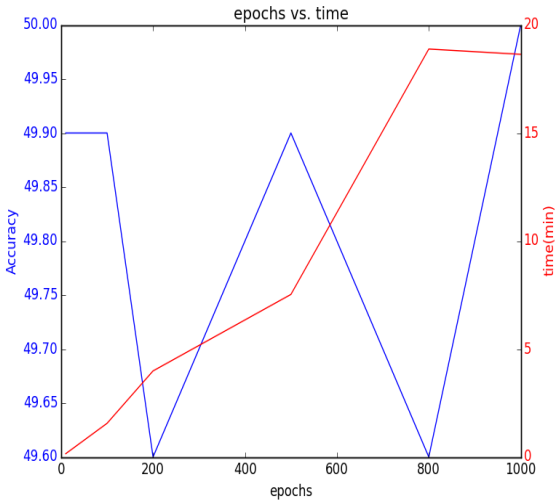


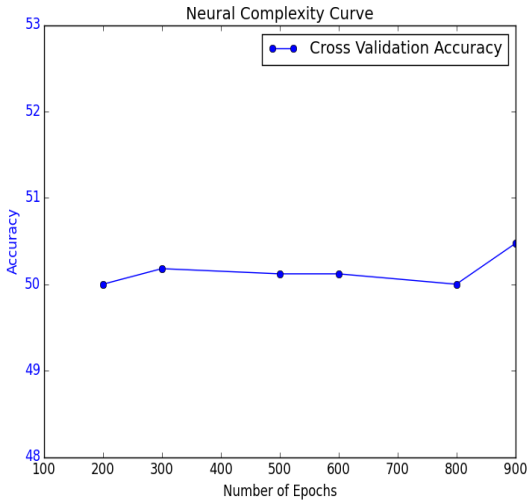
	Training	Testing
Accuracy	0.993	0.68
F1 Score	0.993	0.682
Avg. Precision	0.994	0.715
Avg. Recall	0.996	0.759
Roc Auc score	0.992	0.643

Increasing the number of estimators does not increase the accuracy by a whole lot but, it does increase the training time exponentially. It could be because the model gets over-fitted with the training data which causes the model not to be a good predictor on the testing set.

Neural Network:

MultilevelPerceptron from Weka was used to simulate a neural network to classify tweets. Tweets were split into words and converted to a vector in order to pass it to the perceptron. Due to big size of features the word vector must have been very big because MultilevelPerceptron was not able to get better result than 50%. With each increasing epoch the time taken to build the classifier increases but, accuracy does not increase more than 50%. As we can see in the curve below:





Even, after running for 900 epochs accuracy does not increase and stays around 50% meaning it misclassifies most of the data. It clearly suggests that sentiment classification is not suitable for back propagation on a feed forward network.

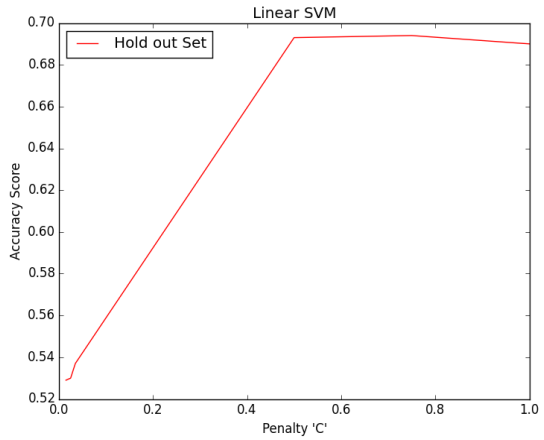
Looking at the confusion matrix for epoch of 1000 we can see that even though the training set contains equal number of positive and negative sentiment tweets. Network is biased towards Negative tweets and classifies each one to be negative.

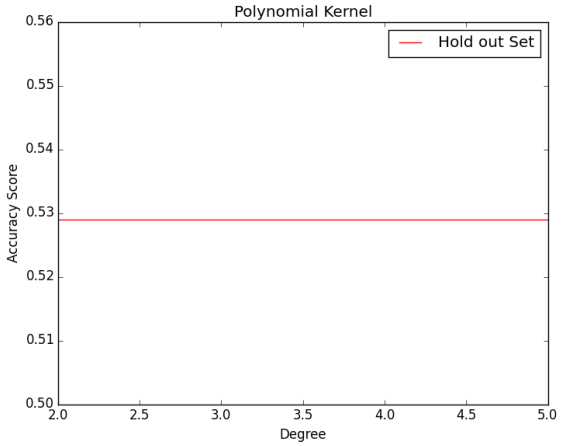
	Negative Predicted	Positive Predicted
True Negative	200	50
True Positive	201	50

Support Vector Machine

Support vector machines with both linear kernel and polynomial kernel was used to analyze the dataset. SVC module from Scikit-library was used to fit and train on the data set. Initially, SVC with a linear kernel was run with default setting which gave an accuracy of 53% on the testing set while polynomial kernel with degree three gave an accuracy of 51% on the same test.

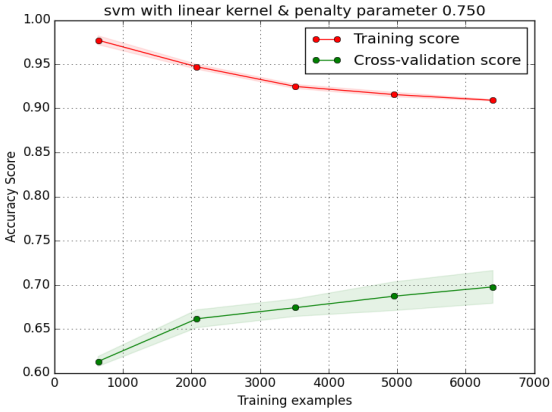
By iterating over the penalty or 'C' parameter model for linear SVC the complexity was analyzed as shown in the graph below. The similar thing was done on the Polynomial kernel but, instead of changing the penalty the degree of the polynomial was changed.

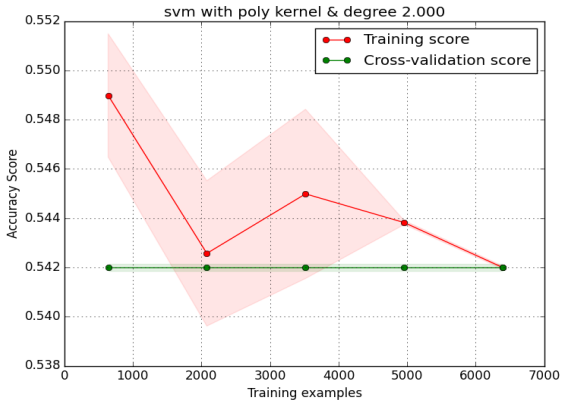




Polynomial kernel SVM gave various accuracy holdout set as the penalty was increased but, as can be seen on the graph changing the degree of the kernel did not change the accuracy of the classifier at all. Linear kernel SVM gave an accuracy of 69% on the holdout set but, Polynomial kernel was at 53%.

Hence, linear kernel with the 'C' parameter of 0.75 was chosen for cross validation and do testing. Polynomial kernel with degree two was chosen to classify the testing set. These two classifiers were then chosen to do cross validation.



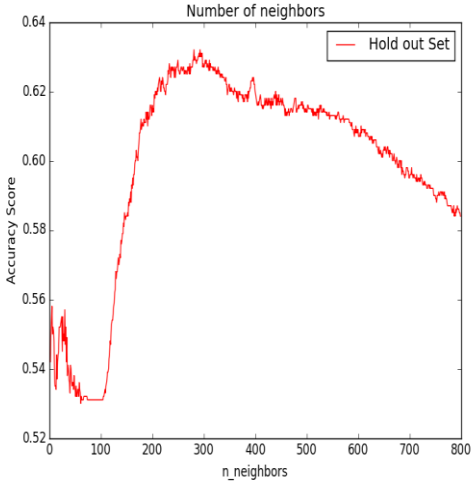


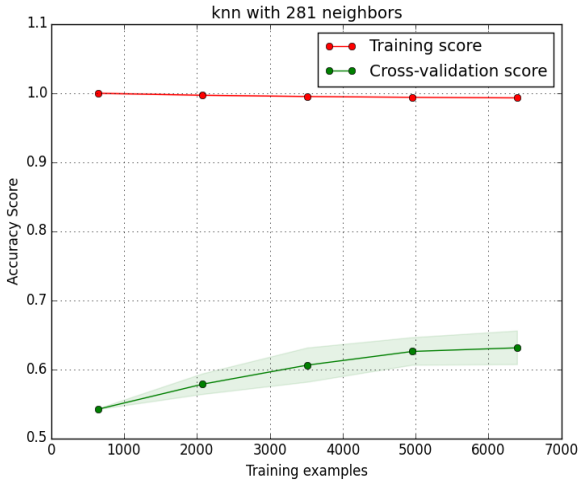
	Linear Training	Linear Testing	Poly Training	Poly Testing
Accuracy	0.993	0.632	0.542	0.529
F1 Score	0.993	0.722	0.703	0.692
Avg. Precision	0.994	0.778	1.00	1.00
Avg. Recall	0.996	0.904	0.771	0.765
Roc Auc score	0.992	0.615	0.500	0.500

The learning graphs show that the dataset is linearly separable also show that the dataset has very low degree because the kernel of degree two also performs much better than higher degree kernels. The linear kernel's testing accuracy can be increased by adding more data.

KNN:

KNeighborsClassifier from Scikit-learn was used to do run the KNN classification. The classifier was trained on the simplest KNeighborsClassifier with the default nearest neighbors of 5. It gave an accuracy of 55% which did not seem promising. So, in order to figure out the optimum number of neighbors training data was fit iteratively and the number of neighbors were increased. The accuracy of the classifier was then compared to the hold-out set. The following fig. _ shows the accuracy against the number of neighbors for the hold out set.





Changing the number of neighbors to 281 gave it the accuracy of 63% on the holdout set. It took 5 mins to iteratively compare the accuracy of KNN for different depth from 5 to 800. As more neighbors are compared bias increases as the testing data tends to get away from the true result which causes the accuracy to go down.

The classifier with 281 neighbors was fit with training data and a cross validation was done and plotted in the figure above. We can see from the chart above that adding more data on training would have improved the accuracy of the classification.

Metrics:

	Training	Testing
Accuracy	0.993	0.632
F1 Score	0.993	0.722
Avg. Precision	0.994	0.778
Avg. Recall	0.996	0.904
Roc Auc score	0.992	0.615

KNN gives similar score to other learning algorithms. It can be improved by giving more data to train.

Conclusion:

Ensemble method is the winner for classifying the tweets into positive and negative group among the ones experimented. It has the highest accuracy of 68%. Only downside to boosting is the amount of time

needed to run the training. Training time increases exponentially to the number of estimators. Decision tree is close second with 64% even though SVM and KNN classifier give similar performance of 63% on the testing set.

This type of data contains huge number of attributes because of various terms appearing on the tweet that are not part of the Standard English. A lot of variation and misspellings for the same word also make it difficult to narrow down the features. For example: 'aaaa', 'aaahhhh', 'aaaw', ‘aaahhaa’ mean something similar to us as we interpret the tweets but, same emotion can be expressed in various forms.

Furthermore, neural network did not perform at all. It could be because Weka only provides back propagation on a perceptron whereas sentiment analysis is better handled with a recurrent network.

Even though, the best accuracy is 68% on the dataset it is pretty good given the dataset with 15000 features. The accuracy also could be improved by doing some natural language processing by comparing synonyms, creating a list of positive and negative attributes and giving them more weight. Another feature that could be added is to convert the emoticons into words like happy, sad, angry etc.

Reference

Agarwal, Apoorv, et al. "Sentiment analysis of twitter data." *Proceedings of the Workshop on Languages in Social Media*. Association for Computational Linguistics, 2011.

Caruana, Rich, and Alexandru Niculescu-Mizil. "Data mining in metric space: an empirical analysis of supervised learning performance criteria." *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004.

Kumar, Akshi, and Teeja Mary Sebastian. "Sentiment analysis on twitter."*IJCSI International Journal of Computer Science Issues* 9.3 (2012): 372-378.

Martínez-Cámara, Eugenio, et al. "Sentiment analysis in twitter." *Natural Language Engineering* 20.01 (2014): 1-28.

Twitter Sentiment Analysis Training Corpus (Dataset). (n.d.). Retrieved September 21, 2015, http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/