Amazon Review Classification Project

1. Introduction

This is the final project for the machine learning nano-degree offered by Udacity.

1.1. Project Overview

The quality of the comments made on internet forums has always suffered due to anonymity of it's users. When users comments on, for instance, a YouTube video, there is no repercussion for what they say, and the dialog generated is often not helpful. Different websites have tried different methods for extracting more useful comments. Reddit uses an up-vote system, Quora fosters a community that values high quality responses over low quality ones, and Amazon allows for its users to to rate the "helpfulness" of reviews left on their products. Amazon's system, in particular, then allows for the higher rated comments to be displayed at the top of the review forum so that new users can see the top rated comments in order to help them make their own purchasing decisions.

Even though Amazon's helpfulness rating system seems to work on the surface level, poor quality comments still seem to be at the top of their review forms. Having poor quality reviews hurts Amazon's business, as a major reason that people are willing to buy consumer goods on-line without seeing the items themselves, is that they have access to others peoples opinions of the item. For example, a review at the top of an app called Friday Night at Freddies 4 is as follows:

> "I love this game so much but at first I though it was lame but when I go in the game I can't beat the first night because cause I put it to full volume and I can't here the breathing bonnie strike at 4 am Chica at 5 and plus it not lame it's better than fnaf and fnaf 2 plus get this game when u buy fnaf"

This comment, despite being at the top of the forum, is difficult to understand, a run on sentence, and full of spelling errors. The reason for the failure is part of the algorithm for determining the order of the reviews relies on how recently the review has been made. The offending review was the most recent, but it's helpfulness score was far less than previous reviews. This illustrates the difficult balance that must be struck between showing the highest rated reviews, and showing the newest reviews, to be rated by the community. An ideal system would predict if a review is helpful or not, so that poor quality reviews would not need to be displayed the top.

1.2. Problem statement

The problem being addressed in this project is the poor quality of Amazon reviews at the top of the forum despite the "helpfulness" rating system. The problem arises from the "free pass" given to

new reviews to be placed at the top of the forum, for a chance to be rated by the community. The proposed solution to this problem is to use machine learning techniques to design a system that “pre rates” new reviews on their “helpfulness” before they are given a position at the top of the forum. This way, poor quality reviews will be more unlikely to be shown at the top of the forum, as they do not get the “freepass” because they are new. The proposed system will use a set of Amazon review data to train itself to predict a helpfulness classification (helpful, or not helpful) for new input data.

The data used for this project is provided by the University of California, San Diego and is available at Julian McAuley's research page. See the references at the end of this project for more information. As the original set is massive, we will only look at a subset of the data: the reviews for apps for Android.

1.3. Metrics

Since our problem is a binary classification problem (helpful or not helpful). We will use the ‘Receiver Operator Characteristic Area Under the Curve’ or `roc_auc score` to evaluate our model. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR). The area under the curve is used to give a score to the model. If the area under the curve is 0.5, then the TPR is equal to the FPR, and the model is doing no better than random guessing. A perfect model would have an AUC of 1.0, meaning it has 100% TPR. For more information on the roc_auc score please visit the sklearn page.

2. Analysis

Section 2 details the analysis that took place on the dataset before any feature generation takes place.

2.1. Data exploration

As a first step, the data is downloaded from the above mentioned site and parsed from ‘JSON’ format to a pandas data frame. Steps to do this are completely contained in the attached iPython notebook. A first look at what the data looks like is shown below:

	reviewerID	asin	reviewerName	helpful	unixReviewTime	reviewText	overall	reviewTime	summary
0	A1N4O8VOJZTDVB	B004A9SDD8	Annette Yancey	[1, 1]	1383350400	Loves the song, so he really couldn't wait to ...	3.0	11 2, 2013	Really cute

The `helpful` score can be explained as follows: a user can either rate the review as 'helpful’ or ‘not helpful’. The dataset records each of these in an array. For our problem we want to classify the email as either ‘helpful’ or not ‘helpful’. For training, this label can be generated by dividing the ‘helpful’ ratings by the total ratings and seeing if it exceeds a certain threshold.

The information used for this project is as follows:

- The 'reviewText' will be used to generate features using natural language processing.
- The score will be used as a feature in our final model. It will not be included in the benchmark model.
- The 'helpful' rating will be used to generate labels. We will train our model using these training labels. Predict the label using the test features, and measure the success of our model using the test labels. This will be explained more later.

Other features that could be important would be the summary; however, in the interest of keeping this simple, we will exclude it from the analysis.

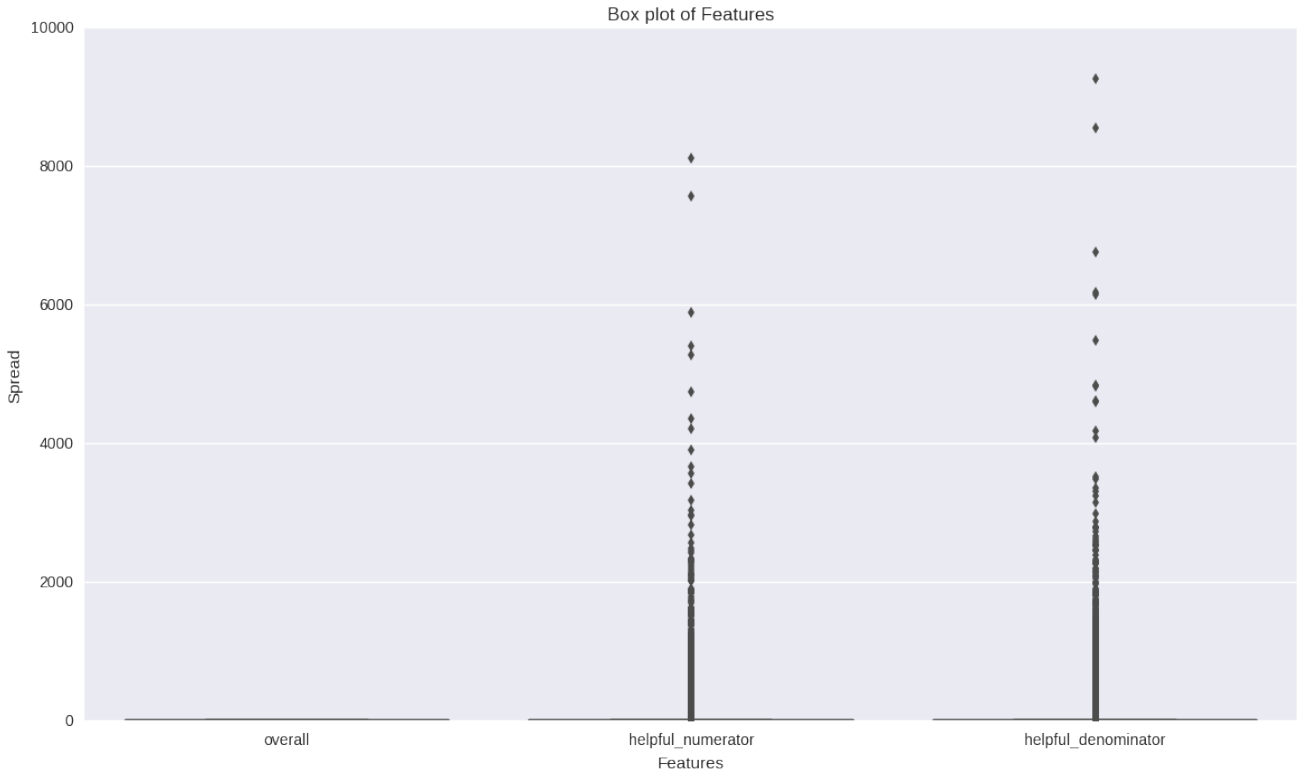
The first things done to look at the data was to determine if there were any missing values. There were none, so all of the data was kept. The second operation performed was to look at some basic statistics. This is shown in the table below:

	overall	helpful_numerator	helpful_denominator
count	752937.000000	752937.000000	752937.000000
mean	3.968931	3.435937	4.473125
std	1.342484	37.328951	43.688824
min	1.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000
50%	5.000000	0.000000	0.000000
75%	5.000000	1.000000	2.000000
max	5.000000	8116.000000	9258.000000

There are clearly some outliers and weird numbers in here. The max for both the helpful numerator and the denominator are both over 1000 standard deviations from the mean of the data.

2.2. Exploratory Visualization

A Box-Plot of the data was produced in order to understand the underlying structure a little better. This visualization is shown below.



As expected there are quite a few outliers. However, these outliers were inspected a found to contain valid text. Since the feature labels will be generated by dividing the helpful_numerator by the helpful_denominator and always producing a value between 0 and 1, these outliers do not really matter for our end model.

3. Methodology

The methodology for pre-processing data and selection a benchmark model will be discussed in the following section.

3.1. Data Preprocessing

Data that has had less than 10 ratings are trimmed out of the dataset. There are a significant amount of reviews, and many do not face the scrutiny of people reading them. As many of these reviews could be good, but by chance, do not get read and rated, they will be rated negatively by our algorithm and will cause the model to incorrectly classify future "good" reviews.

As it is desired to determine if a given review text is helpful or not, a way to map the existing data to this binary classification is needed. The chosen method is to use a threshold of 'helpful_numerator' divided by 'helpful_denominator'. In other words, the ratio of the people who found the review helpful over the amount of people who rated the review as helpful or unhelpful. If this ratio exceeds a certain threshold value, we can label the training data as 'helpful' = 1, or 'non-helpful' = 0. For this analysis,

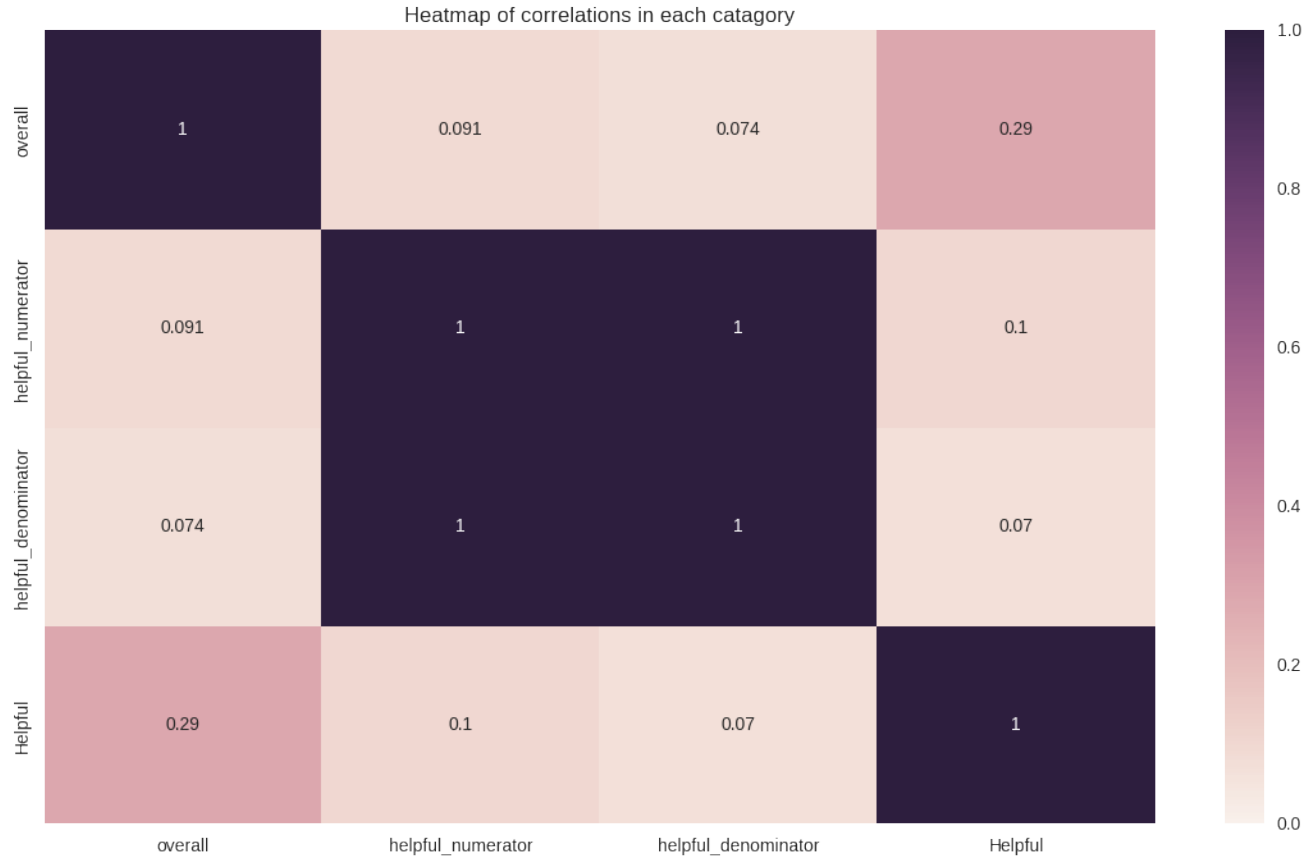
the threshold is arbitrarily set to 0.5, however, this could be tuned in the future to a higher value in order to better filter the reviews.

The final count of each of the classification after performing this operation is shown in the following table:

	overall	reviewText	helpful_numerator	helpful_denominator
Helpful				
0	6210	6210	6210	6210
1	41722	41722	41722	41722

It can be seen that the dataset in unbalanced, with roughly 7 times as many effective reviews as ineffective reviews.

The correlation of the variables can be visualized with the following heat map:



There is a small correlation between the score and the 'Helpful' rating. It is good that this correlation is small, or else the job would be very easy and we wouldn't need to generate features from the reviewText at all, we could just look at the score of the review.

3.2. Text Feature Generation for Benchmark Model

The text is converted to lower case to run natural language processing to produce features. Also removed is punctuation. Removing punctuation may seem like it should not be done in the case of this problem, as not having punctuation will make a review harder to understand. This is a trade-off that must be considered. If all punctuation is conserved it will cause the learning algorithm to behave poorly.

In order to generate more features the TF-IDF vectorizer from the sci-kit learn library is used. The pre-processing methods that were used are as follows:

- ***Stemming*** - Stemming means to take off the suffixes of the stemmed word. Therefore words such as "run" and "running" would both be represented as "run". This allows for our algorithms to more accurate find trends in the "meanings" of sentences (e.g "the sun is shining" and "the sun shines" will more accurately be associated with the same meaning. It will reduce the total amount of features that we will generate.
- ***Tolkenizing***- Splits sentences up into single words. This is needed to generate our TFIDF features.
- ***Remove Stop Word*** - This moves words such as "the" "a" and "it" as shown in the English stop words. Admittingly, sentences without these words are hard to understand, and ideally they should be kept, but it is again a trade-off.
- ***Ngrams*** - Makes groups of words that are 'n' long. E.g the 2-grams for the sentence "The shaggy dog" are [The, shaggy], [The, dog] and [shaggy, dog]. Having more than one is computationally expensive as it increase the word count by n chose 2.

Finally TF-IDF scores were generated for each of the stemmed and tolkenized words and ngrams. TF-IDF is short for Term Frequency Inverse Document frequency. TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. The TF-IDF score can be explained as follows.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

The TF-IDF weight is the product of these two numbers. For more information check out this http://www.tfidf.com/

For the vectorizer, the min_df was set to 0.001. This is the minimum document frequency (I.e the percentage of documents that the word appears in). This is a cheap way of getting rid of spelling mistakes. Words that appear this infrequently are most likely typos and are uninteresting to our algorithm. It also reduces the feature space, allowing the algorithm to work faster.

3.3. Implementation (Benchmark Model)

In order to establish a baseline for the project, the follow 'out-of-box-models' from sklearn were examined:

- Gaussian Naive Bayes (GaussianNB)
- Decision Trees
- Ensemble Methods (Bagging, AdaBoost, Random Forest)
- Logistic Regression

Information on each algorithm can be found at http://scikit-learn.org/stable/supervised_learning.html.

Taking this documentation into consideration, the following three algorithms were selected to be investigated, for reasons listed below:

Logistic Regression: Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier.

- *Real World Application:* Logistic Regression is used frequently for text classification, and is popular in Kaggle competitions as a baseline classifier.
- *Strengths of Model:* Logistic regression can be updated after is has already been trained, meaning that new reviews can be used to teach the algorithm after it has already been training.
- *Weaknesses of Model:* Logistic regression is not as fast as the naive bayes methods, but then again, not many algorithms are.
- *Selection Justification*: The ability to update itself after its initial training period may prove to be valuable.

Random Forest: A Random Forest is a ensembling algorithm that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. It does not implement boosting on the trees, and instead counts a vote of each individual tree in order to produce the final class label.

- *Real World Application:* Random Forest has been used in Diabetic Retinopathy classification analyses (5), and is very popular for Kaggle competitions.
- *Strengths of Model:* The strength of the Random Forest classifier comes from the formation of it's trees. Because it is formed from "random" subsets of the data, and the final result is compared to other trees that have also been formed "randomly", the algorithm guards well against "overfitting" from noisy data points that may have more influence on a single decision tree algorithm. The "random" formation of the trees ensures that there is little chance for a strong bias to be present in the data during tree construction. (4) Random Forests also work well with high dimensional data, as in the case of our TFIDF features.(5).
- *Weaknesses of Model:* A lot of trees are necessary to get stable estimates of variable importance and proximity. This can lead to a large amount of space in memory being needed to store the trees. Additionally, the trees need to be re-trained when new data is being introduced, unlike Naive Bayes. Its training complexity is given as $O(M \sqrt{d}\, n \log n)$ where d is the number of features and M is the number of trees.
- *Selection Justification:* Since random forest works well with high dimensional data, as is competitive with other algorithms such as SVM without the high training cost (5) it is selected for our study.

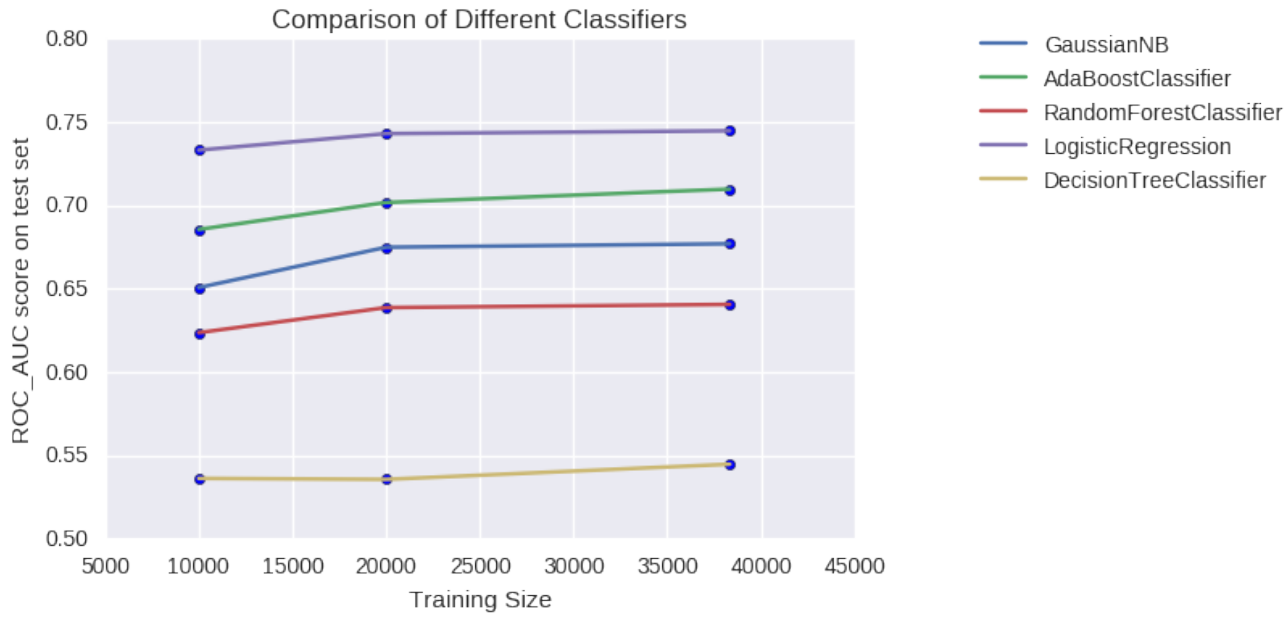
Adaboost: AdaBoost or "adaptive boosting" begins by fitting a "weak" classifier on the original dataset. It then fits additional copies of the classifier on the same dataset and adjusts the weights of incorrectly classified instances such that subsequent classifiers focus more on difficult cases. The adjustment is done using the SAMME-R algorithm (for classification)

- *Real World Application:* Adaboost has been used in robust real time face detection (6). It is also quite popular for Kaggle competitions.
- *Strengths of Model:* Because of it's boosting property, Adaboost will not suffer from overfitting caused by too many training periods in the boosting algorithm. Theoretically, the algorithm should produce better and better results the more it is trained.
- *Weaknesses of Model:* Adaboost is slow to train, relative to GaussianNB (actual training complexity could not be found).
- *Selection Justification:* It will be interesting to measure the performance of the boosting method of Adaboost vs the vote method for the RF algorithm, as both use decision trees and differ only in the method in which the tree are ensembled together for the final classification.

The data was then shuffled and split into *X_train, y_train, X_test and y_test* sets. The benefit to splitting the data into testing and training sets is that this allows simulated evaluation of how well the model is performing before using it in the real world to make predictions. If there was no testing set, there would be no way of realistically evaluating the model as you would only be able to measure it's

performance on data to which it's already been exposed. This will result in a false sense of confidence in how well the model performs.

Finally each of the algorithms listed above were tested using the train/test split, and three different sizes of data (10000,20000,38345). The results can be summarized by the following plot:



The Logistic Regression is the best algorithm in terms of accuracy for all test sizes. It's final score for the area under the ROC curve was 0.7448 and a sample size of ~38,000. In addition it is the fastest in terms of training and prediction. The training speed and prediction speed were 0.538s and 0.0137s respectively . As our system needs to consider the trade off between accuracy and speed, the Logistic Regression algorithm represents the ideal model for our benchmark. Surprisingly, the Random Forest algorithm did quite poorly in our tests. It was both slower and less accurate than the Naive Bayes algorithm.

3.4. Refinement

In order to improve the results a new feature was added to the feature set. The 'overall' score category was shown before to have a correlation with 'helpful' or 0.29. It was added to the feature set.

As before, the data was split into *X_train, y_train, x_test and y_test.* In addition, an optimization was performed on the model using the following techniques:

3.5. Grid Search

The grid search technique works by generating a grid of all possible provided parameter combinations. It then evaluates a model using a validation set based on every combination of

parameters in the grid. It is used to find the optimum set of parameters for a learning algorithm given a data set.

3.6. Cross-Validation

The k-folds validation training technique creates multiple testing and training sets and trains a model on each, averaging the results. The splitting method is to divide the data into separate bins (for example k = 5), train on bins 1 to k-1 and test on bin k. The next fold trains on bins 2 to k and tests on bin 1 and so on, until all of the bins have acted as a test bin. This effectively trains and tests the models on all of the data without over-fitting to the data.

The benefits that Cross-validation provides for grid search while optimizing the model is that there is less of a chance that the final model won't be optimized to data that could potentially have a bias. For example, if somehow, all of the training data didn't contain a unique data point (such as a high MEDV value for a low RM) the final model would be very biased not predict this scenario, and the optimization will make that worse. However, with k-folds validation, one of the trained models will see the unique data point and the result will be reflected in the final averaged model.

The optimized model has the following parameters:

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

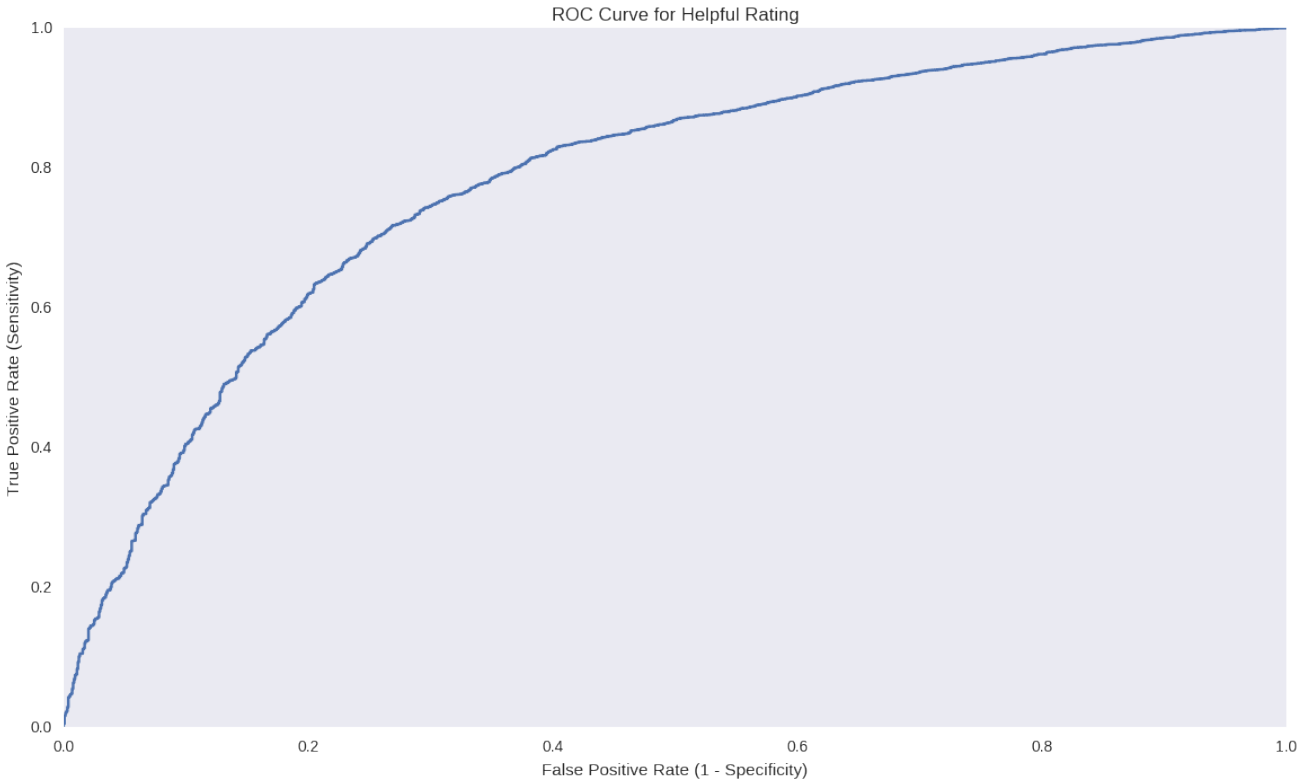
These parameters are the default parameters of the Logistic Regression model from Sklearn.

4. Results

The results of our optimization are discussed in the following section.

4.1. Model Evaluation and Validation

The Radio Operator Curve for the optimized model with the ‘overall’ score added is visualized in the following plot:



The final area under the ROC score for our optimized model is 0.7785.

4.2. Justification

There are several variables can be examined in order to determine the robustness of the solution. THe feature space size was significantly reduced by setting the min_df (minimum document frequency) to 0.01, meaning that the terms needed to appear in the documents at least 1 percent of the time. This was done as an effort to eliminate spelling errors, but it also may have eliminated important words. For this project this was not examined, as increasing the amount of features is beyond the available computational capabilities.

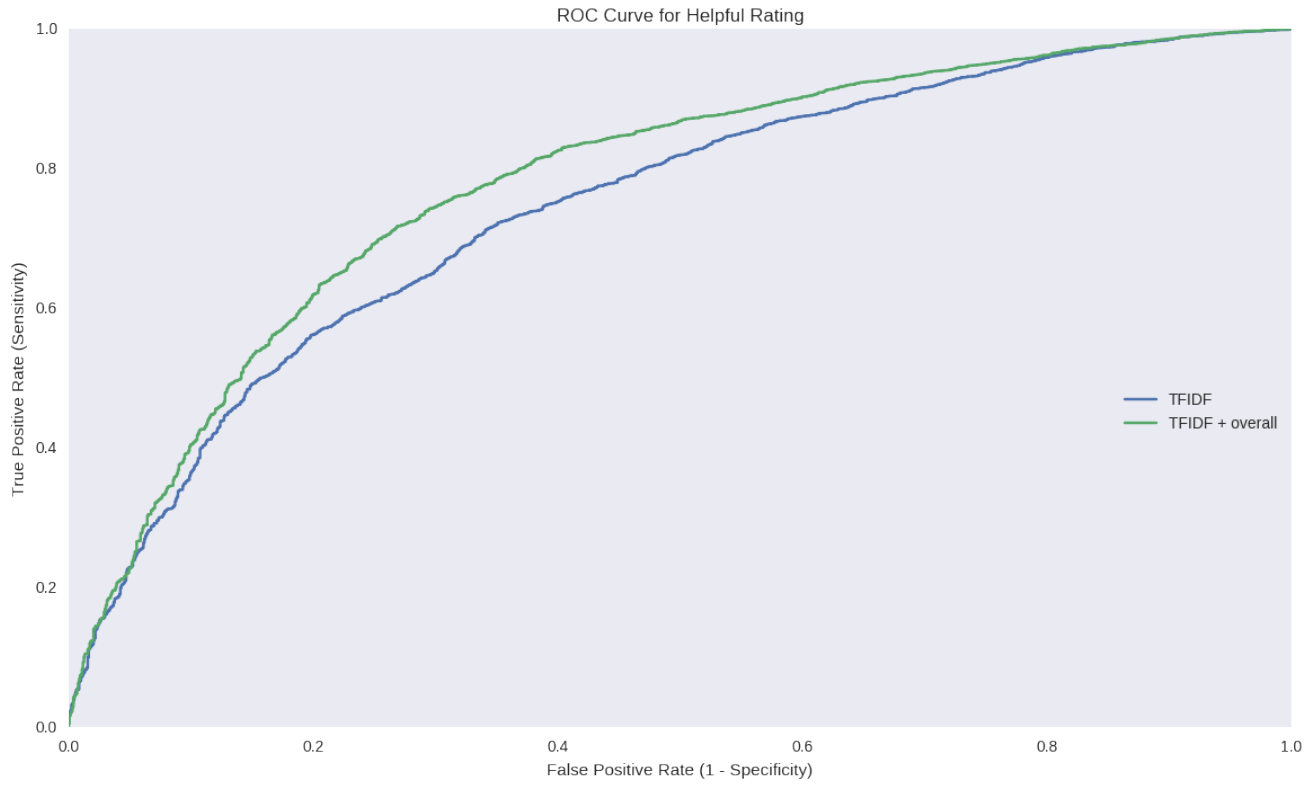
Another variable that could be examined is the random_state variable of the Logistic Regression algorithm itself. The random_state variable for the Logistic Regression algorithm controls the seed of the pseudo random number generator to use when shuffling the data. If this number is changed and the auc_roc score of our classifier changes significantly, that could speak to the robustness of the solution.

In order to investigate the effect of changing the random seed for the Logistic Regression model, an experiment was set up to run the algorithm with 100 different random seeds from 1-100. The average value of the resulting auc_roc score was 0.778, which is consistent with the results obtained in the previous trials. So in this regard, our results can be considered robust.

5. Conclusion

5.1. Free-Form Visualization

A important quality of this project is the effect of introducing a new key feature to a benchmark model. Mainly, we looked at the TFIDF features generated from amazon review text and added the 'overall_rating' that was given to the product by the reviewer. We used these features to predict how 'helpful' other users would find the review. Our results can be visualized as follows:



Clearly seen in the ROC curve is the result of adding the 'overall' score to the feature set. The Area under the curve for the features set of TFIDF features plus to 'overall' score is notably higher.

5.2. Improvements

In order to improve the accuracy of our model, the following actions could be taken:

- Perform more feature engineering, such as proper spell checking for the reviews. This would result in a model that potentially has less features, as certain spelling errors would have been corrected/eliminated.
- Explore the effect of not doing pre-processing on the reviews. Reviews with poor grammar, and punctuation and improper word endings are more difficult to understand and would possibly lead to people rating them as less helpful. During our project, we did this pre-processing in

order to make the algorithms work better; however, maybe feature reduction should have been done in a different way to order to preserve the "incorrect" information.

5.3. Reflection

A summary of the problem solution is as follows: THe model takes in existing 'reviewText' and transforms it into numerical TF-IDF scores. It then adds the existing 'overall' score of the reviews to create a features set for each review. It trains a Logistic Regression model using labels generated by taking existing 'helpfulness_numerator' data and dividing it by 'helpfulness_denominator' data and thresholding the result at 0.5. It then takes in new reviews with no helpfulness data from user ratings and attempts to classify them as being 'helpful' or 'non-helpful'. By using this system, Amazon can work to make sure than more 'helpful' reviews are shown at the top of their forums.

This project is interesting in the sense that it is not like typical text classification problems. In many text classification problems, you are trying to reduce the dimensionality of text data as much as possible by combining synonyms, stemming words, and correcting spelling mistakes. However, in the context of this problem, that may have not been desirable as then main reason that people may have found a review unhelpful is because of these imperfections. It would be interesting to explore this problem again, without doing some of these preprocessing steps.

Another interesting thing to try would be to extract the most 'useful' words from the TF-IDF feature set. This could be done using the `.vocabulary_` attribute from the TF-IDF vectorizer and the `feature_importance_` attribute from the `LogisticRegression()` classifier. The result of this analysis could be used to generate Review Guide to help new reviewers write better reviews.

6. References

[1] Inferring networks of substitutable and complementary products. J. McAuley, R. Pandey, J. Leskovec Knowledge Discovery and Data Mining, 2015.

[2] Image-based recommendations on styles and substitutes J. McAuley, C. Targett, J. Shi, A. van den Hengel SIGIR, 2015.

[3] HUI BWU Y. Anti-spam model based on semi-Naive Bayesian classification model. Journal of Computer Applications. 2009;29(3):903-904.

[4] Liaw A. Weiner M. Classification and Regression by randomForest. R News. 2002;Vol 2(2):18-22.

[5] Casanova R, Saldana S, Chew EY, Danis RP, Greven CM, et al. (2014) Application of Random Forests Methods to Diabetic Retinopathy Classification Analyses. PLoS ONE 9(6): e98587. doi: 10.1371/journal.pone.0098587

[6] Jones M. Viola A. Robust Real-Time Face Detection. International Journal of Computer Vision. 2004. pg 137–154.

[7] http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/