**Capstone Project**

Ross Beck-MacNeil  
November 22nd, 2017

**1 - Definition**

**1.1 - Project Overview**

Machine learning has been successfully used to classify documents by topic for several decades[[1]](#footnote-1). However, machine learning techniques do not perform as well when performing sentiment analysis which requires the parsing of more complex language structures[[2]](#footnote-2). This is where ideas from natural language processing must be applied. Humour is even more subjective and is harder to analyze than sentiment.

My aim with this project is to make progress towards solving a complex natural language processing problem using machine learning. I think that it would very beneficial if computers could interpret and produce the same kind of natural language of which even young children are capable. This would allow machine learning to be applied to a wider variety of tasks than it is currently capable of solving. In particular, machine learning could be applied to many problems that are not well structured and for which there is not much training data.

**1.2 - Problem Statement**

I will attempt to solve the problem of determining whether a post on the reddit r\jokes subreddit has received more than 10 net upvotes, conditional on its title and submission text. The criteria of 10 net upvotes can be considered a proxy for the true metric of interest, the funniness of the post. However, humor is very subjective and hard to quantify and therefore the net upvotes from the r/jokes subreddit is being used as a proxy for the funniness of the post.

This is a problem that reoccurs every hour of every day, as users are constantly posting new submission to r/jokes. Not only is the r/jokes subreddit constantly producing a new stream of problems to be solved, there is ample historical data so that a model can be trained and learned.

This will be a supervised binary classification problem and as such many different measurable evaluation metrics could be applied. These will be further elaborated in *Section 6 - Evaluation Metrics*. Given the title and text of a post, the solution will be a machine learning model that receives as input the text and title of the post and outputs either a class prediction or a class probability. The solution will be further described in *Section 4 - Solution Statement.*

**1.3 - Metrics**

Given that this is binary classification problem, there are many different metrics that could be used to evaluate the solution. The most common metric is accuracy. If is the true value of observation i, such that

.

and is the predicted value of the *i*-th observation, then the accuracy of the predicted values can be calculated as:

where *N* is the number of samples and is the indicator function such that:

However, the accuracy metric can be misleading in certain circumstances. For one, it can be hard to interpret accuracy when one class occurs much more frequently. This is known as an unbalanced classification problem. In addition, it does not account for the confidence that the classifier has in its predictions. It seems reasonable to prefer a model that is more confident in its correct predictions and less confident in its incorrect predictions.

The Area Under the Receiver Operator Curve (AUROC) metric can overcome the shortcomings of the accuracy metric. The Receiver Operator Characteristic (ROC) is a graphical plot that illustrates the classification ability of a binary model as its discrimination threshold is varied. The area under this curve (AUC) is the probability that the classifier will produce a more confident positive prediction for a randomly chosen positive example than a randomly chosen negative example.

The one drawback to the AUROC is that it can only be calculated for algorithms that output confidence scores. Some algorithms such as support vector machines and k-nearest neighbors do not natively output such confidence scores. Therefore, this project will focus on models that natively produce confidence scores.

It is important to emphasis that proposed solution will be ultimately evaluated using its predictions on the hold-out test set. It is quite easy to develop a classifier that achieves perfect accuracy or AUROC on data that it has seen. It is quite another task to develop a classifier that does well on unseen data. However, care must be taken not to evaluate too many potential solutions against the hold-out set. Ideally model and hyperparametric selection will take place using cross-fold validation or a validation set that is different from the test set. Using the test set for these purposes could lead to overfitting in the same way as using to directly optimize the model parameters

**2 - Analysis**

**2.1 - Data Exploration**

**2.1.1 – Reddit Jokes**

The primary dataset that is being used contains all posts from the Reddit subreddit r/jokes that were submitted between January 25, 2008 and September 10th, 2017. I personally obtained this dataset by using the reddit API to query and retrieve these submissions.

The input dataset is in comma separated values (csv) format and is entitled *AllSubsFrom\_rJokes\_14\_09\_2017.csv*. The 14\_09\_2017 indicates the date that the scraping finished: September 14th, 2017. The data set, before any cleaning or processing, contains 319,747 observations. Each observation corresponds to a different reddit post. There are a total of 9 variables in this data set and they are described in Table 1.

Table

|  |  |
| --- | --- |
| **Variable** | **Description** |
| *id* | Unique identifier assigned by Reddit |
| *date* | Post submission date, in Unix epoch time (seconds since January 1, 1970) |
| *downs* | Number of downvotes. Due to reddit protections, always equal to 0 |
| *score* | Net upvotes (fuzzed) |
| *text* | Text of submission/post |
| *title* | Title of submission/post |
| *ups* | Number of upvoter. Due to reddit protections, this is equal to the score variable |
| *upvote\_ratio* | Upvote ratio, ratio of upvotes to total votes |
| *url* | Link to original post. |

Given the scope of the problem, it is very appropriate to use this data set, as it conforms exactly to the problem statement from section 2. Per the specifications of the problem statement, a new *funny* variable will be created such that for *ith* observation:

Jokes with 1 < score < 10 are considered neutral, neither funny nor unfunny, and will not be considered by the model and will dropped from the dataset. This approach is similar to the one employed by Bo et al (2002) where they focus on discriminating between positive and negative movie reviews and do not consider neutral ones. This *funny* variable will be the target variable that the solution will attempt to predict. At times it may be referred to as *y*.

After removing neutral posts, 12.5% of the observations will be randomly selected into a holdout test set. This will the set of observations upon which the final model will be evaluated. The models will be developed only against the remaining 87.5% of observations. This separation between the train and test sets is very important.

Table 2 shows the five “funniest” jokes, as indicated by the number of upvotes. Note that the fifth joke does not contain any text. This is likely because is submission that contains a link rather than text. Given that the machine learning model that I will be developing will require text, this is a problem. The solution is detailed in *Section 3.1 – Preprocessing*.

**Table 2 – Top Five Submissions**

|  |  |  |
| --- | --- | --- |
| **Number of Upvotes** | **Title** | **Text** |
| 98086 | V | V    \*Edit: seems like the ctrl key on my keyboard is not working |
| 90293 | The 2016 US Presidential Election | That's it. That's the entire fucking joke. |
| 85380 | Did you hear about the Doctor on the United Flight? | [removed] |
| 73522 | This is the dirty joke my 85yo grandad told to our whole family by memory | A male whale and a female whale were swimming off the coast of Japan when they noticed a whaling ship. The male whale recognized it as the same ship that had harpooned his father many years earlier. He said to the female whale, "Lets both swim under the ship and blow out of our air holes at the same time and it should cause the ship to turn over and sink." They tried it and sure enough, the ship turned over and quickly sank.     Soon however, the whales realized the sailors had jumped overboard and were swimming to the safety of shore. The male was enraged that they were going to get away and told the female, "Let's swim after them and gobble them up before they reach the shore." At this point, he realized the female was becoming reluctant to follow him. "Look," she said, "I went along with the blow job, but I absolutely refuse to swallow the seamen."     Edit: I think it's bad that I'm more excited watching this get ups that I was about the whole of Christmas |
| 66971 | The funniest /r/jokes has ever been |  |

**2.1.2 – Pretrained Glove Embeddings**

In addition to the dataset containing the reddit posts, datasets consisting of pretrained GloVe word embeddings will used. These datasets are available from <https://nlp.stanford.edu/projects/glove/> (Pennington et al. 2014). Each dataset contains one line per word, where each line starts with the word and is followed the elements of its vector representation. The word and its elements are separated by spaces. In order to test the effect of using vector embeddings of different dimensions, **two** of these pre-trained word embeddings files will be used. The data sets that will be used will correspond to vectors of 50 and 300 dimensions. The 50 dimensional vectors are the smallest, while the 300 are the largest. They will provide a good contrast. It is expected that the pre-trained embeddings will allow for the development of models that account for semantic similarities while avoiding overfitting to the relatively small training set.

**2.1.3 - Exploratory Visualization**

Figure 1 below, supports the decision to turn this into a binary classification problem rather than treating it as a regression problem. It shows that the distribution of scores is very uneven; the 99% percentile only accounts for 34% of total upvotes. This means that the top 1% of posts account for over 65% of total upvotes.

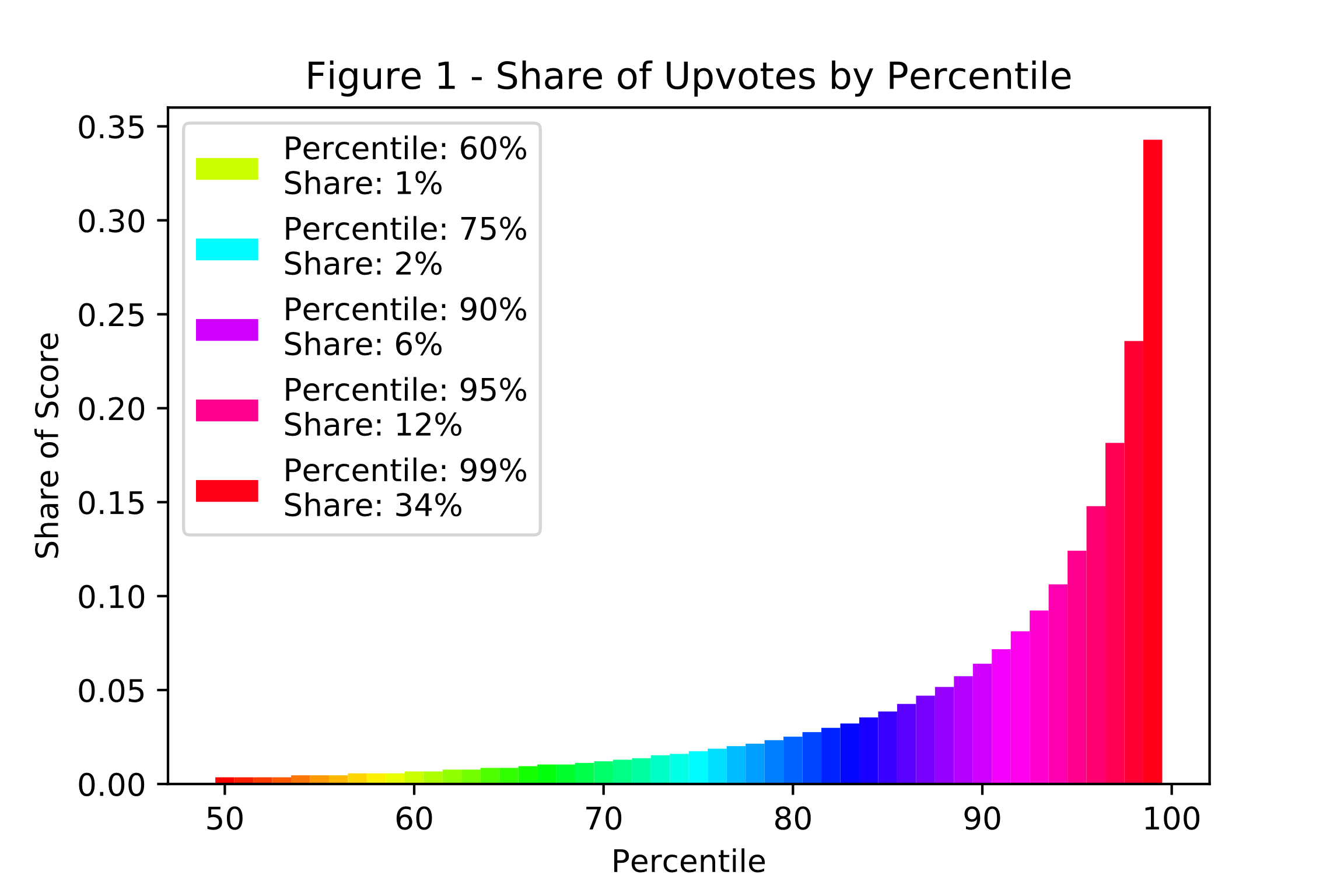
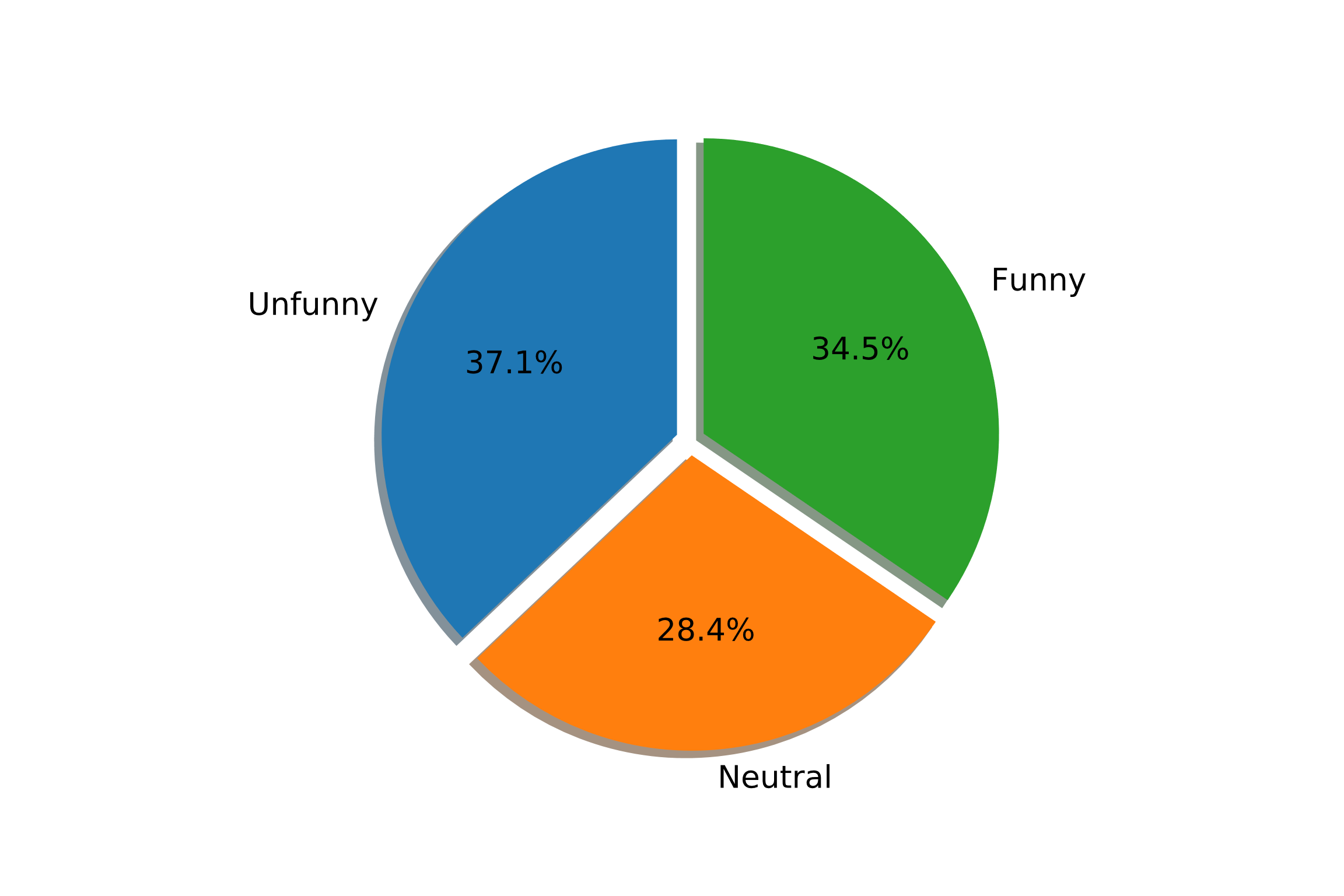
****

Figure 2 below, supports the decision to classify posts as funny if score greater than equal to 10, unfunny if 0 or 1, and to drop if between 1 and 10. Very even breakdown.

Figure 2 – Distribution of Jokes by Type



**2.2 - Algorithms and Techniques**

I will be employing three types of algorithms to solve this problem: linear models, decision trees and neural networks. The linear logistic and decision tree models are “traditional” machine learning models and will act upon the same transformed data. The data will be processed differently for the neural network models.

The linear logistic regression model will form my benchmark. Linear logistic regression is a good benchmark because it is a simple model that does not require the tuning of many hyperparameters. It is also fast to train and robust to noisy data (Source?). The expectation that more “advanced” techniques should achieve superior performance. Otherwise, what is the point? The benchmark linear model achieves an accuracy of **68.44% and an Area Under the Curve of 0.7443 on the test set.**

Decision trees can learn non-linear relationships that a linear logistic regression cannot. They can also be trained on the same type of input data as the logistic regression. Unfortunately, decision trees are prone to overfitting. In order to overcome this limitation, addition to using a single decision tree model, I will use the random forest model. The random forest model takes the decision tree and decreases variance by adding in randomness.

Finally, will turn to the new hot thing, neural networks. Neural networks are very flexible. Can do many things. Good thing about them is that the take in data in different way from decision trees and and decision trees. As will be discussed in preprocessing section, can learn word order.

# **3 - Methodology**

### 3.1 - Data Preprocessing

### As indicated in Section 2.1 “Data Exploration”, the raw data required some cleaning and transformation. The following steps were followed:

1. Observations with “text” variables of three or less characters were dropped
2. The data was transformed into a binary classification problem by:
   1. Dropping observations with between 2 and 9 upvotes (inclusive)
   2. Creating a binary variable “funny” that equaled zero if “ups” <= 1, and one otherwise
3. Concatenating the “title” and “text” variables into a single “full\_text” variable
4. Splitting the data into a test and train, using a 87.5%/12.5% split. This resulted in
   1. A train data set with 196509 observations
   2. A test data set with 28073 observations

**3.1.1 – Bag of Words Model**

The bag-of-words model was used to create the input data set that was consumed by the linear logistic regression and decision tree models. The bag of words model tracks word occurrence but discards word order. I used the sklearn’s TfidfVectorizer class to produce a document term matrix. In a document term matrix, every row represents a document (joke) while every column represents a term (feature). The elements of the document term matrix correspond to the occurrence of a term in a particular document. The “Tfidf” in TfidfVectorizer stands “term-frequency inverse document frequency”. This means that within a document more frequent terms are accorded a higher weight, while terms that appear in many documents are accorded less weight.

The following parameters were provided for the TfidfVectorizer instance:

|  |  |  |
| --- | --- | --- |
| **Argument** | **Value** | **Explanation** |
| *analyzer* | ‘word’ | Sklearn built in tokenizer. “The default regexp selects tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator).” |
| stop\_words | 'english' | Common words such as “the” and “a” are removed |
| *min\_df* | 5 | Only retain terms that appear in at least 5 train documents |
| *max\_df* | 0.8 | Remove common terms that appear in at least 80% of the train records |
| *ngram\_range* | (1,3) | Create n-grams of 1 to 3 words. |
| *max\_features* | 231850 | Only include the consider top max\_features terms, ordered by term frequency. Corresponds to all of the terms (after accounting for tother filter teps such as min\_df, and soptowkrds). This value was ontained though cross vlaidotn,. Dicusedd further later. |
| lowercase | True | Convert all characters to lowercase. Consierably reduices number of features. |

**3.1.2 – Dense Word Embeddings**

The neural network models require that the data be processed differently. The benefit of the neural networks is that they can take word order into account. To do this

The neural network model uses the pretrained Glove embeddings. The Glove embeddings are dense representations of words.

In addition to labels, the

For neural network, the following preprocessing steps were followed

1. Create vocabulary
   1. Sklearn count vectorizer
   2. Nlktk word\_punct\_tooeknizer
2. Embeddings
3. Masking
4. Min

**3.2 - Implementation**

The LogisticRegressionCV classifier from the sklearn package was used to develop the benchmark linear model. The CV in “LogisticRegressionCV” stands for cross-fold validation. This is because this model searches through a range of “C” values in an efficient manner, while validating the performance of the model with a given C value using cross fold validation. C values are how sklearn specifies the regularization strength for linear models. A low value for C implies high regularization, meaning that the coefficients are penalized when they far from 0.

Best C value and number of features found using **8 fold** cross validation – grid search (accuracy). 8 values of C were searched. From 0.01 to 100.0, spaced evenly in log space. It was found that C value of 1.930698 gave best results. In fact, it seemed that this was more important than max)features in dtm. Top four linear models had C value of 1.930698.

As mentioned in section 3.1.1, the development of the benchmark linear model was also used as an opportunity to determine the optimal number of features to retain in the document term matrix. I tried 8 different values for *max\_features*, from 28981 to 231850 in an even linear fashion. Corresponding to 12.5% to 100% of maximum number of terms (when no restriction). As mentioned in section 3.1.1, it was determined that max\_featues of 231850 gave best results. This means no addiontal filtering.

**3.2.2 – Decision Trees**

The decision tree models were also developed with the sklearn package. For the models consisting of a single decision tree, the sklearn’s DecisionTreeClassifier was used. For random forest models, sklearn’s RandomForestClassifier was used. Like with LogisticRegressionCV, the fit method can be called on the train document term matrix in order to train the model after initializing the classes with the desired hyperparemets. Section 3.3 refinement goes into more details about how these hyper paramters were specified.

**Complications?**

**Grid searching for decision tree**

Adding estimators and oob error for random forest. Random forest takes a fair while to train.

**3.2.3 Neural Networks**

The neural network models were developed with the Keras package using the TensorFlow backend. Keras provides the API to Tensorflow. Tensorflow ran on my GPU. This sped up computations significantly.

See figure \_\_ for model achtiructure.

Explantion:

|  |  |
| --- | --- |
| Layer Name | Explanation |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* **Recurrent neural network**
* Neural networks
* Embedding layer
* Recurrent layer
* Dense
* Dense (output

**Complications**

Masking, development of averaging layer, overfitting, long fit times…

Regularization….

**3.3 – Refinement**

The, untuned single decision tree that used sklearn’s default parameters achieved an accuracy of 61.71% and a AUC of .6150 on the test set. These are very poor results compared to the benchmark linear model. Given that the untuned decision tree model obtained an accuracy of 98.553% on the train set, it is clear that the untuned decision tree is overfitting to the train data. This overfitting is likely harming its ability generalize and thus its performance on the test set. To reduce this overfitting and to hopefully obtain superior results, I tried several different combinations of two different hyperparameters that control for overfitting. The following table shows these hyperparameters:

In particular, two DecisionTreeClassifier parameters were modified from their default values. These were:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Definition[[3]](#footnote-3)** | **Default** |
| *min\_sample\_split* | The minimum number of samples required to be at a leaf node | 2 |
| min\_impurity\_decrease | A node will be split if this split induces a decrease of the impurity greater than or equal to this value. | 0 |

Different values for the above two parameters were tried. Specifically, values equal to or higher than their defaults were tried, since higher values for these parameters imply more regularization and consequently less overfitting. A randomized search on these hyperparameters was tried, using sklearn’s RandomizedSearchCV class. Values for *min\_sample\_split* were sampled uniformly from a list containing the integers 2-14 (inclusive). Values for *min\_impurity\_decrease* were sampled from a uniform distribution with a minimum of 0 and a maximum of 0.0005. The RandomizedSearchCV instance randomly generated 100 different combinations of these parameters and evaluated the performance of each combination using 5-fold cross validation. This means that overall the RandomizedSearchCV instance created 500 different models.

After completing this search process, the best combination of parameters was found to be *min\_impurity\_decrease***:** 0.000028 and *min\_samples\_split*: 12. This set of hyperparameters obtained a cross validation accuracy of 62.55%. On the test set, the model trained using these hyperparameters obtained an accuracy of 63.149% and an AUC of 0.67629. This equates to small improvement compared to the untuned decision tree model. Despite this small improvement, the tuned decision tree model still underperforms the benchmark linear model by a significant margin. It is clear that the decision tree is no longer overfitting, given that it does not obtain an accuracy of better than 67.260% on the train data.

**Random Forest**

For the random forest model, I did not train a model using sklearn’s default parameters. Instead, I first trained a model with 50 estimators (trees) instead of sklearn’s default of 10. I validated the performance of this model using out of bag error. It achieved:

Sqrt: 0.675267

Log2: 0.670270

The small random forest models (with 50 estimators/trees) had the worst out of bag performance. I then added more tree/estimators in increments of 50, up to 600. Each time recording oob accuracy.

The model with the best out of error had the most estimaotrr/trees: 600. It use log2 to select features. It achieved an oob socre of 0.698436. It was therefore tested on the test set. On the test set it achieve an accuracy of 69.554376% and a AUC of 0.7549.

**4 - Results**

**4.1 - Model Evaluation and Validation**

As indicated in section 3.3, the best performing model was the random forest model that was trained with 600 trees and looked at log2 features each split. This model achieved an out-of-bag accuracy of 69.84%. On the the test This is very similar to the accuracy of 69.55% that this model achieved on the test set. It also achieved a AUC of 0.7549 on the test set.

It is not surprising that best random forest model has most estimators. Performance of the random forest model increases with number of estimators.

Feature importance for random forest.

It is possible that the random forest model is suffering from overfitting, or at least close to it. I base this observation on the fact that

**4.2 – Justification**

The random forest does not really beat benchmark linear model. It is true that it obtains slightly higher accuracy and area under the curve. However the random forest requires much more time to train, prediction time, size of model. Also bigger difference between accuracy of test and train…

Compare to benchmark linear model with accuracy of **68.44% and an Area Under the Curve of 0.7443 on the test set.** This is an improvement of 1.11% in accuracy and 0.0106 in AUC.

RF takes 1 minute 34 secs to predict 28073 test obserrvations, and takes \_\_ to predict 196509 train observations. Final model has size of 8 gigabytes.

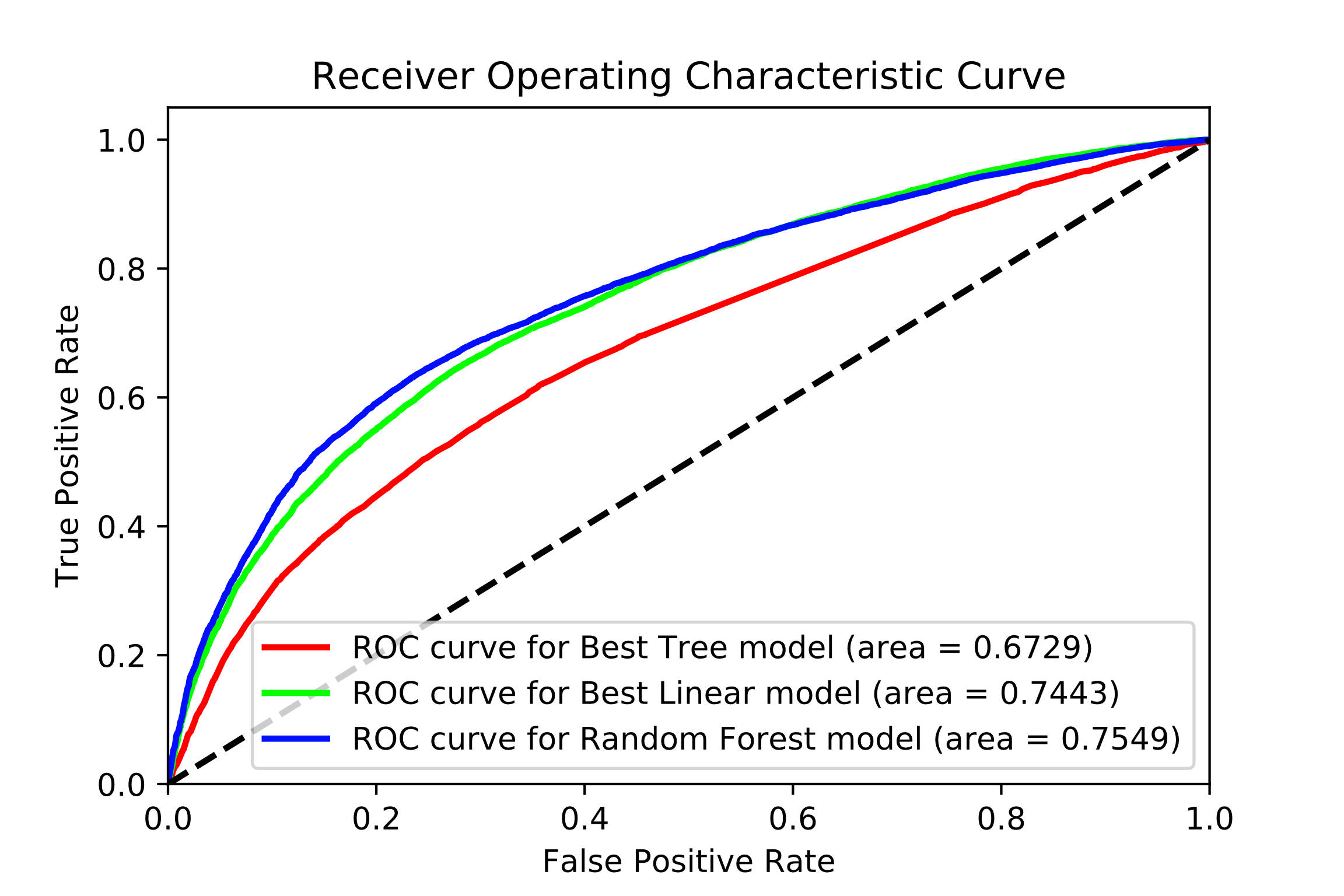
Compare to linear model that only takes 56 seconds to fit to the model. Quite difference. Also linear model is less than 2 megabytes in size.

In this case, I think I should declare that the benchmark linear model that I have developed has not been beaten.

**5 - Conclusion**

**5.1 - Free-Form Visualization**

The following figure shows the ROC for the best performing models of the different types. Again, one can observe that



**5.2 - Reflection**

Interesting that it is so hard to beat linear model. That is surprising.

Porably reflective of the fact that there is realtivel limited amount of training fdaata

**Improvement**

* **Get more data**
* **Joke generation**

**References**

1. Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques." *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002.
2. Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

1. Sebastiani, Fabrizio. "Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1 (2002): 1-47 [↑](#footnote-ref-1)
2. (Bo et al. 2002). [↑](#footnote-ref-2)
3. http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html [↑](#footnote-ref-3)