**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 submissions 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 1.3 - 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.

**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 1

|  |  |
| --- | --- |
| **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 it is submission that contains a link rather than text. This is problematic given that I will be developing machine learning models that require text. 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/> [[3]](#footnote-3). 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. To test the effect of using vector embeddings of different dimensions, two of these pre-trained word embeddings files will be used, containing vectors of 50 and 300 dimensions. The 50-dimensional vectors are the smallest available, while the 300-dimensional vectors are the largest. It will be interesting to compare and contrast models trained on the largest vectors with models trained on the smallest vectors. 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.

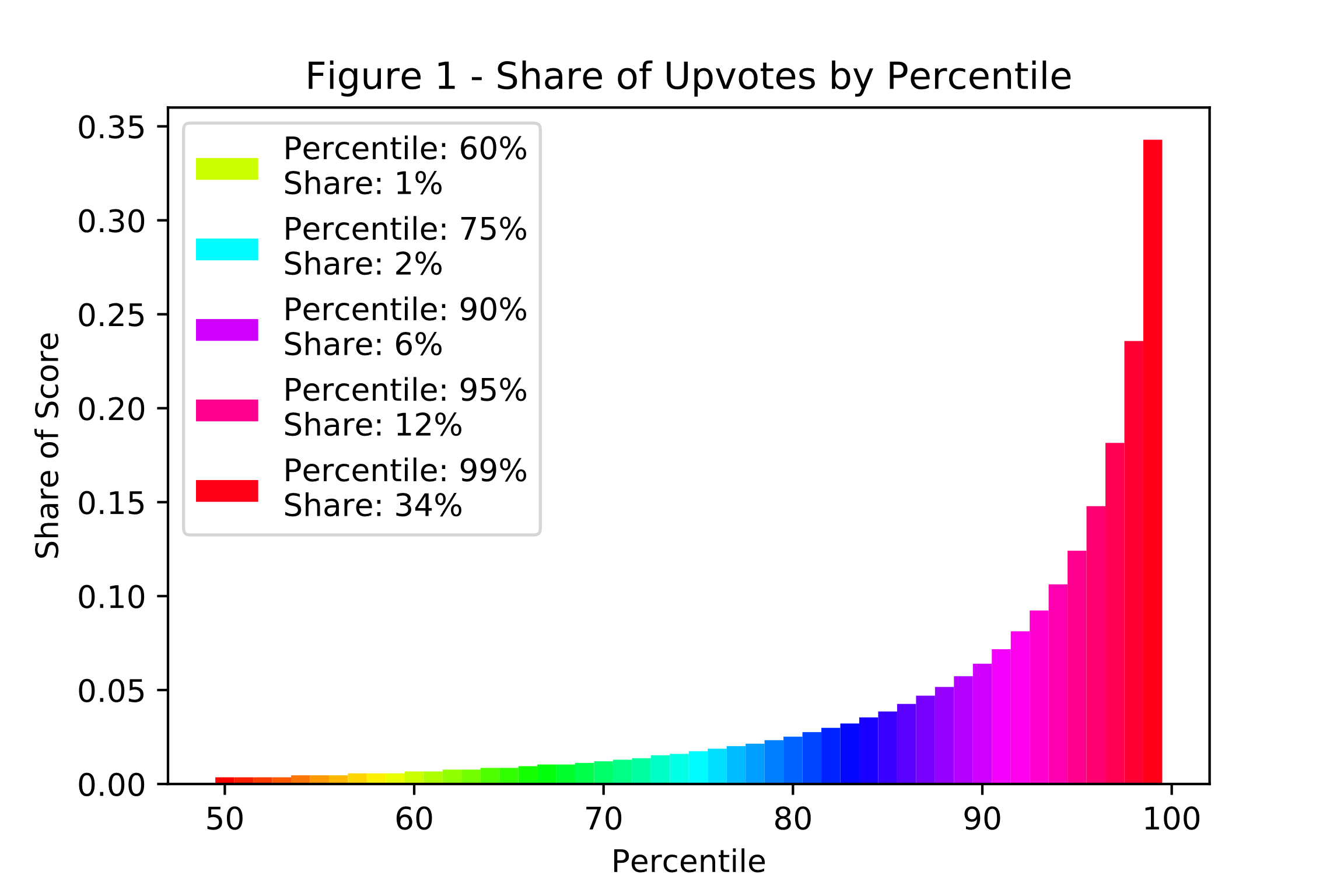
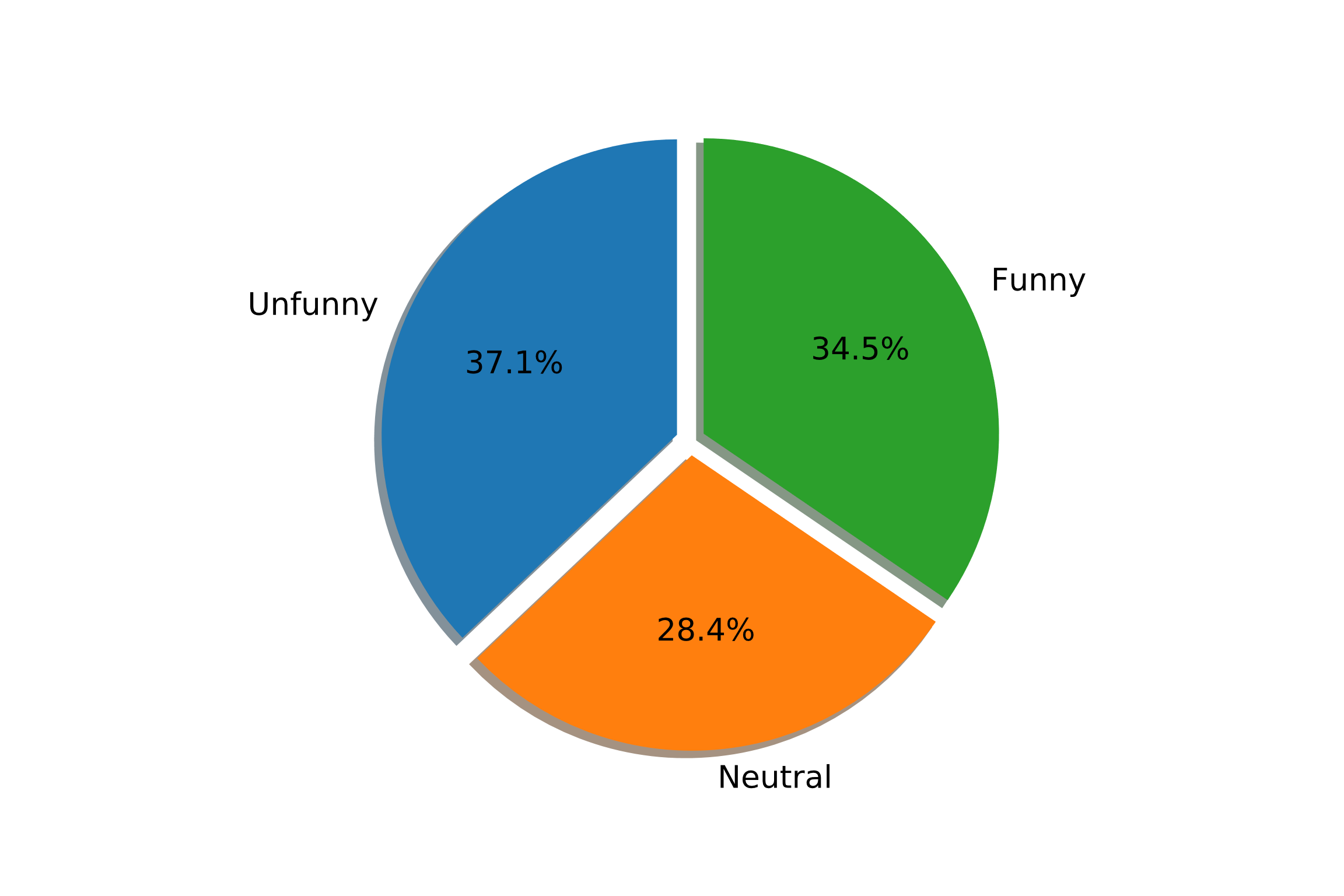
****

Figure 2, below, validates the manner by which jokes labeled. It shows that the labeling leads to a very even split.

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 input data. The neural network models will require that the data be pre-processed in a different manner.

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 is that the more complex decision tree and neural network models should achieve superior performance. The benchmark linear model achieves an accuracy of **68.44% and an AUC 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, in addition to using a single decision tree model, I will use the random forest model[[4]](#footnote-4). The random forest model trains several decision tree models independently and averages their results. Bagging of observations and consideration of a reduced number of features at each split are employed to decrease variance by adding in randomness.

Neural networks are even more flexible than decision trees. They can do many things. The good thing about them is that the take in data in different way from linear logistic regression and decision trees. As will be discussed in preprocessing section, can learn word order. The drawback to neural networks is that they can be slow to train and have very many hyperparameters.

# **3 - Methodology**

### 3.1 - Data Preprocessing

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

1. Observations with a *text* variable 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 196,509 observations
   2. A test data set with 28,073 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[[5]](#footnote-5) 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 for “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 a weight lower. The TfidfVectorizer class was instantiated with the following parameters:

|  |  |  |
| --- | --- | --- |
| **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. The value of 231850 corresponds to keeping all of the terms, after accounting for other filtering steps such as *min\_df*, and *stopwords*. This value was obtained through a grid search that employed cross validation. Discussed further later. |
| lowercase | True | Convert all characters to lowercase. Considerably reduces number of features. |

**3.1.2 – Dense Word Embeddings**

An advantage of neural networks over other machine learning algorithms is that it is possible for a neural network to take word order into account. However, this means that the input data cannot be structured using the bag of words model. Instead, every joke is represented as a vector of a fixed length; these vector representations are also called sequences. The *ith* element of this vector corresponds to the *ith*word in the joke’s text. Each word is represented as an integer that corresponds to a row in an embedding matrix. This embedding matrix will be used during training to transform these single dimensional sequences of integers into two dimensional matrices. The rows of these matrices will correspond to a word, with the columns providing a multidimensional representation of the word. As already discussed, I will be employing 50 and 300 dimensional embeddings. These embeddings will be pretrained using the GloVe algorithm, rather than being initialized using random weights. The use of pretrained embedding takes prior knowledge into account and should help with training. I chose the sequences a fixed length of 300, because percentiles. Please see below for order of steps.

1. The sklearn CountVectorizer class used to create a vocabulary that assigned words to indexes
   1. The NLTK[[6]](#footnote-6) packages’s *wordpunct\_tokenize* function was used in place of sklearn’s default tokenizer
2. The embedding matrix was created per the following steps:
   1. The pre-trained GloVe word vectors were loaded from disk into a dictionary
   2. A random matrix was created
      1. Number of rows = length of vocabulary (number of words)
      2. Number of columns = dimensions of word vectors
   3. For every word in vocabulary, the corresponding row of the embedding matrix was updated
      1. if there was no corresponding pretrained word vector, the word’s entry in the embedding matrix was left blank
3. Sequences
   1. Sequences were created by splitting sentence into lists of tokes using same tokenizer as was use for the CountVectorizer. Then corresponding index was looked up in the vocab from step 1. Then list was transformed into an aray of ixed length of 300. 0 padding vlaus used at end if shorter than 300 truncrteed if longer.
4. Created validation set from train sequnces., 12.5%.

**3.2 – Implementation**

## 3.2.1 - Benchmark 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 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 model is penalized for having coefficients that are from 0.

The best C value and number of features was found using 8-fold cross validation grid search. This means that I tried 8 values of C, from 0.01 to 100.0 spaced evenly in log space. The performance of each C value was evaluated using the accuracy cross-validation accuracy. It was found that C value of 1.9307 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.

In addition to searching for the optimal C values, 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, as mentioned in section 3.1.1. 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 Tree Implementation**

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

**3.2.3 Neural Networks**

The neural network models were developed using the Keras[[7]](#footnote-7) package frontend to the TensorFlow backend. The Tensorflow computations were executed on my GPU in order to obtain significant speed ups. The main component of the neural network model was a recurrent neural network layer. The recurrent layer treated each word in sequence, like temporal data. Recurrent neural networks incorporate long term dependencies. This is good for text, that is read from beginning to end. Figure 4 provides details on the model architecture. It can be read from top to bottom, following the arrows. Each level is a layer of the neural network. The cells on the left give the name of layer, as well as the name of the Keras layer class that was used for that particular layer. The cells on the right provide the dimension of the inputs and outputs of that layer. The first *None* in the brackets indicates that is dependent on batch size (so corresponds to number of observations in mini batch). Table 3 provides detail on each layer of the model architecture. *Section 3.3.3* explains how the architecture was varied slightly. The neural network was trained with the Adam optimizer using the default settings.. Throughout the training process, the AUC of the validation set was monitored and training was stopped once AUC had not improved for 25 epochs. This monitoring of the AUC required the creation of custom Callback class in Keras. The model was only saved once the AUC of the validation set had improved, thus ensuring that only the best iteration of the model was kept.

|  |  |  |
| --- | --- | --- |
| Table 3 | | Figure 3 |
| **Layer Name** | **Explanation** |  |
| joke\_seq | The input data as sequences,. |
| Embedding | The pretrained embedding matrix is applied to the sequences to transform them to dense vectors. The *trainable* parameter is set to false, so that the matrix will not be updated during training. |
| mask\_paddings | When all values of a word are 0, skip in all subsequent layers. Since the padding are assigned vectors of all 0, they will be masked. |
| drop\_words | Apply dropout[[8]](#footnote-8) to the words by randomly setting percentage (15%) of words to 0 during training. Helps prevent overfitting. |
| mask\_dropped\_words | Mask dropped words, so that they are not considered by the subsequent recurrent layer. |
| reccurent\_layer | The Long-Short Term Memory[[9]](#footnote-9) (LSTM) variant of a recurrent network. Outputs a vector of 150 tanh, between -1 and 1 (tanh activation function) |
| drop\_dense | Apply dropout to 30% of thee recurrent layer outputs |
| dense\_sigmoid | A fully connected layer with 75 hidden units with the sigmoid activation function |
| avg\_pred | Custom layer in Keras. Take average of previous layer to produce output. Averaging like this is a form of regularization |

**3.3 – Refinement**

#### 3.3.1 Single Decision Tree Refinement

The single, untuned, 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 98.55% on the train set, it is clear that the untuned decision tree overfits 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. In particular, two DecisionTreeClassifier parameters were modified from their default values. These were:

|  |  |  |
| --- | --- | --- |
| Table 4 | | |
| **Parameter** | **Definition[[10]](#footnote-10)** | **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 *min\_sample\_split* and *min\_impurity\_decrease* 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 run, 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 63.15% and an AUC of 0.6763. 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. At least the tuned decision tree is no longer overfitting, given that it obtains an accuracy of only than 67.26% on the train data.

#### 3.3.2 Random Forest Refinement

|  |  |
| --- | --- |
| Table 5 | |
| **Max\_features** | **Number of features considered per split** |
| Sqrt | sqrt(231850) = 481 |
| Log2 | log2(231850) = 17 |

I did not train an initial random forest model with sklearn’s default parameters. Instead, I first trained a model with 50 trees rather than sklearn’s default of 10. I then iteratively added more trees to this random forest model in increments of 50, up to a maximum of 600. Each time after adding more trees, I recorded the model’s out-of-bag accuracy[[11]](#footnote-11). I followed this process for two different values of the *max\_features* hyperparameter: “sqrt” and “log2”. The *max\_features* parameter controls how many different features are considered at each split, since the random forest model randomly considers a subset of features for each split. Table 5 shows how these values correspond to the number of features.

Figure 4 displays the relationship between out-of-bag accuracy and the number of trees for the two different sequence of models that were constructed. With only 50 trees, Sqrt obtained an accuracy of 67.527% and Log2 obtained an accuracy of 67.027%. I then added more trees to the random forest, 50 at a time. These initial models were the worst performing. The small random forest models (with 50 estimators/trees) had the worst out of bag performanceThe model with the best out of error had the most trees, 600. It uses log2 to select features. It achieved an oob score of 0.698436. It was therefore tested on the test set. On the test set it achieve an accuracy of 69.55% and a AUC of 0.7549.

#### 3.3.3 Neural Network Refinement

In addition to the initial neural network model that outlined in Section 3.2.3, I trained an additional three models that differed slightly. The first modified model used 300-dimension word embeddings instead of the 50-dimensional embeddings. This modification was discussed in *Section 3.2.3 Neural Network Preprocessing*. In addition to increasing the size of the embedding matrix, I increased the dropout rate that was applied to the inputs of the recurrent layer from 30% to 45% to help prevent overfitting. Otherwise everything else remained the same.

In addition to this modification, I developed two models that were allowed to update the weights of the embedding layer. I did not randomly initialize the weights of these two models. Instead, I used the weights that I had obtained after training the first two models where the embedding layer had been kept fixed. I did this since I did not want to adjust embedding layer too much and I wanted to start from known good values. When training embedding layer, I increased dropout rate that was applied to the inputs of the recurrent layer. Table \_\_ shows the

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Embedding Dimension** | **Trainable Embedding?** | **Trainable Parameters** | **Patience** | **Learning rate** | **Input Dropout Rate** | **Best Validation AUC** |
| 50 | False | 131,925 | 25 | 0.001 | 30% | 73.13% |
| 300 | False | 281,925 | 25 | 0.001 | 40% | 74.87% |
| 50 | True | 1,501,725 | 50 | 0.00025 | 45% |  |
| 300 | True | 8,500,725 | 50 | 0.00025 | 60% |  |

**4 - Results**

**4.1 - Model Evaluation and Validation**

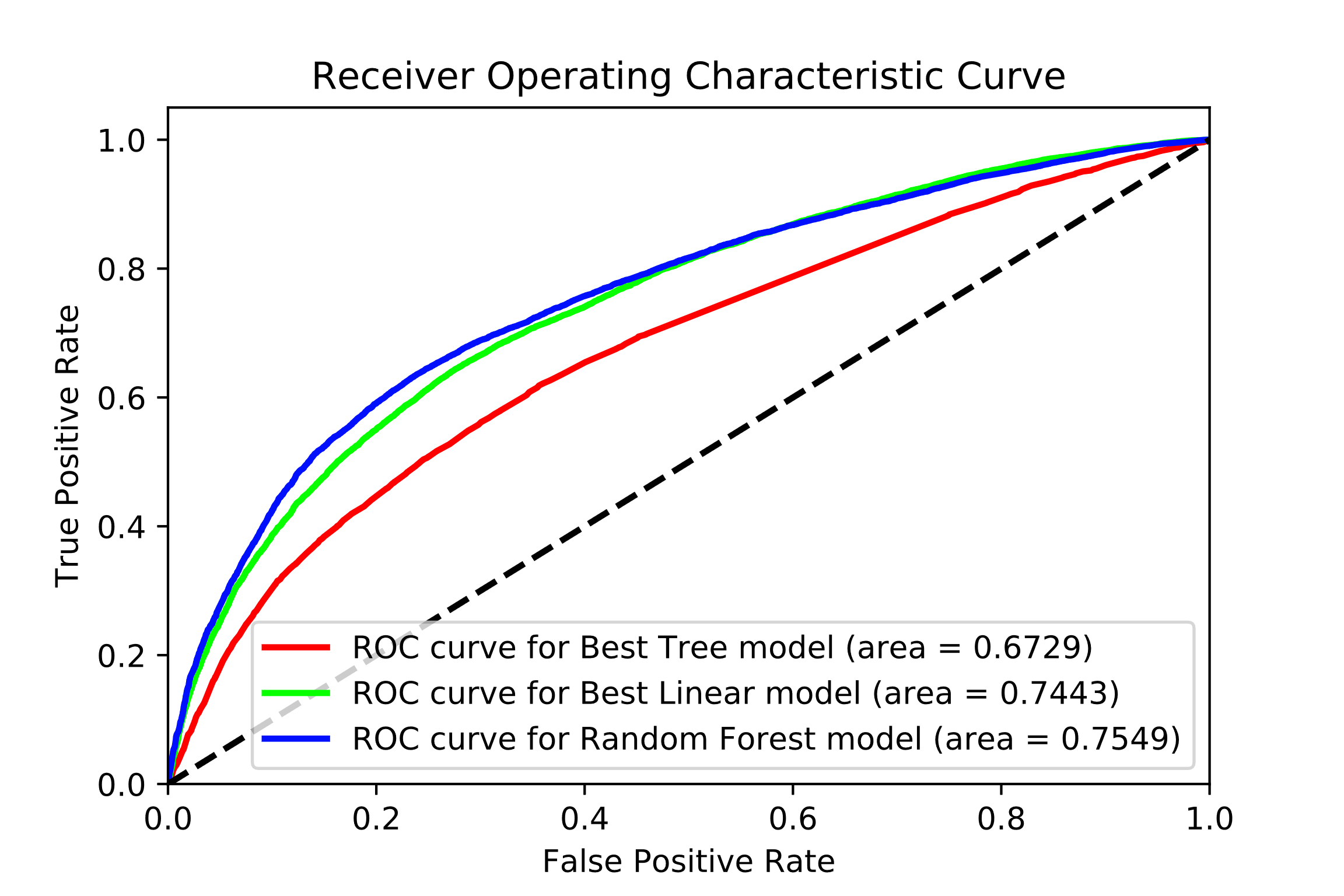
As indicated in section 3.3, the best performing model was the random forest model that was trained with 600 trees and looked at log2 features each split. This model achieved an out-of-bag accuracy of 69.84%. On the the test This is very similar to the accuracy of 69.55% that this model achieved on the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

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

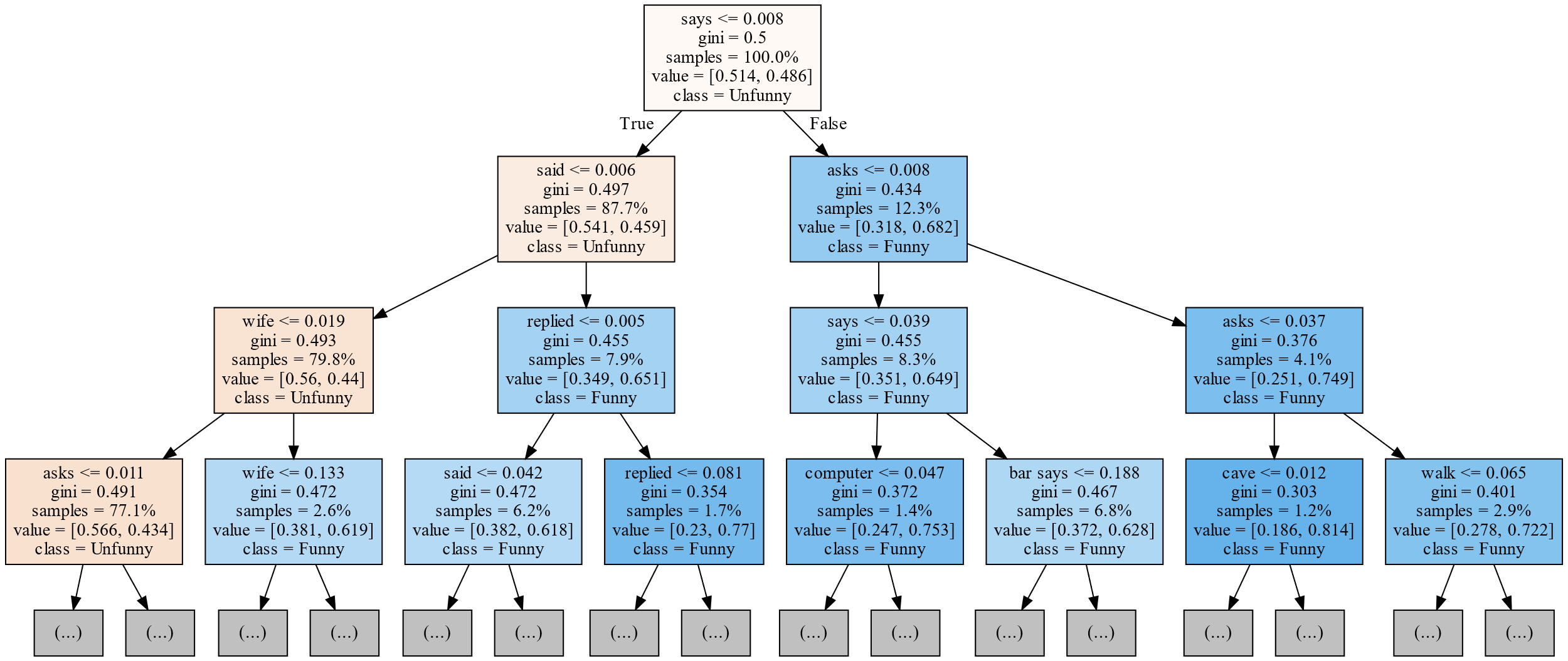
**5 - Conclusion**

**5.1 - Free-Form Visualization**

Figure 3 shows the first three layers of the decision tree model trained using the default sklearn parameters. Decision trees are good because easy to visualize. In this case can’t visualize very much, but oh well.

Shows a lot of similar words said and says both near top, asks, replied

Figure 3 – Decision Tree Visualization



**5.2 - Reflection**

This project developed several different model to tackle the problem of classifying reddit posts as funny or not. This was meant to be a proxy for humor in general, so avoided adding extra meta data such as author of post or date. The models that were spanned a range of complexity that went from simple logistic regression to more complex recurrent neural networks.

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

Probably reflective of the fact that there is relatively limited amount of training data. Also humour is subjective. Also very noisy data. Upvotes can be due to amny things othr than text of submission such sas time of submission and topical humor, meometmum etc…

**5.3 Improvement**

The next thing that I would want to do with this project is to transition to the generation of jokes. To do this, I would need to use a neural network. The random forst and benchmark liner models would not be of any use for thisl Hiepiugflly neural network vlaissifr cvan generalize in some way. Hwoever would alos wanyt to hget more data to learn grammer and string together idea. More dta would nnot hjave to be jokes could be just about any text wit h proper stucutre. Howver would want to not be too far fromm domain of jokes. Would probably want short sotrirs, ideally funny or hunorus hsort stories. Maybe there is shouce UDnno.

Alternaive improve: Make Reddit but that takes this calssifer and makes preoictions, positng on kokes r.subredit as comment. Might not ber allowed by sureddit rules….

**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.

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. 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. [↑](#footnote-ref-3)
4. L. Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001. [↑](#footnote-ref-4)
5. [Scikit-learn: Machine Learning in Python](http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html), Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011. [↑](#footnote-ref-5)
6. Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O’Reilly Media Inc. [↑](#footnote-ref-6)
7. Chollet, Francois, et al. "Keras." *Github* https://github.com/fchollet/keras (2015). [↑](#footnote-ref-7)
8. Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research* 15.1 (2014): 1929-1958. [↑](#footnote-ref-8)
9. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780. [↑](#footnote-ref-9)
10. http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html [↑](#footnote-ref-10)
11. “An important feature of random forest is its use of *out of-bag (OOB)* samples*: For each observation zi = (xi, yi), construct its random forest predictor by averaging* only *those trees corresponding to boot-strap samples in which zi* did not *appear.*” Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc. [↑](#footnote-ref-11)