CAPSTONE PROJECT

Ross Beck-MacNeil
December 18th, 2017

1 - Definition

1.1 - Project Overview

Machine learning has been successfully used to classify documents by topic for several decades[1]. However, machine learning techniques do not perform as well when performing sentiment analysis which requires the parsing of more complex language structures[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[3] 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] Sebastiani, Fabrizio. "Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1 (2002): 1-47

[2] 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.

[3] https://about.reddit.com/

1.3 - Metrics

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

$funny_i \epsilon 0,1\ \{1,0\}$ and $\widehat{funny}_i$is the predicted value of the *i*-th observation, then the accuracy of the predicted values can be calculated as:

$$accuracy(funny, \widehat{funny}) = \frac{1}{N}\sum_{i=1}^{N} 1(funny_i = \widehat{funny}_i)$$

where *N* is the number of samples and $1(funny_i = \widehat{funny}_i)$ is the indicator function such that:

$$1(funny_i = \widehat{funny}_i) := \begin{cases} 1\ if\ funny_i = \widehat{funny}_i \\ 0\ if\ funny_{\ i} \neq \widehat{funny}_i \end{cases}$$

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 metric 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 the performance of the final solution will ultimately evaluated against a hold-out test set. It is quite easy to develop a classifier that achieves perfect accuracy or AUROC on data that it has already seen. It is quite another task to develop a classifier that generalizes well to unseen data. However, care must be taken not to evaluate too many potential solutions against the hold-out test set. The performance of models with different hyperparameters will be evaluated using cross-fold validation or a validation set before being evaluated on the hold out test. The use of the test set for hyperparameter selection could lead to overfitting in much the same the same way as using it to directly optimize the model parameters.

2 - Analysis

2.1 - Data Exploration

2.1.1 – Reddit Jokes

The primary dataset 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.

Per the specifications of the problem statement, a new *funny* variable will be created such that for *ith* observation:

$$funny_i := \begin{cases} 1 \; if \; score_i \geq 10 \\ 0 \; if \; score_i \leq 1 \end{cases}$$

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/ [4]. 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.

[4] 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.

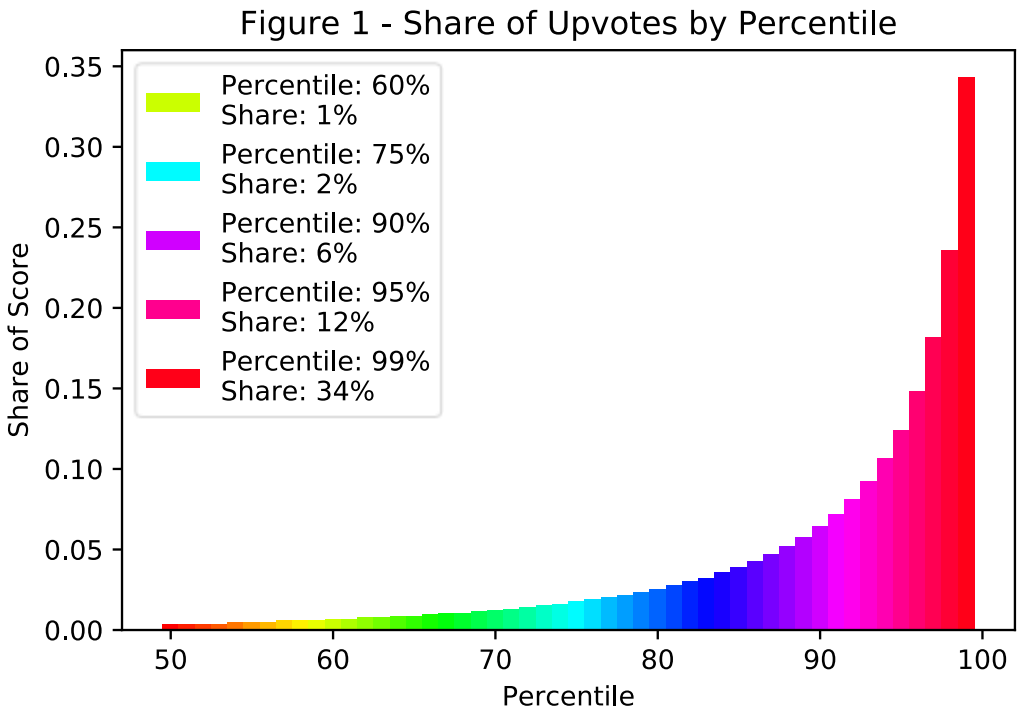
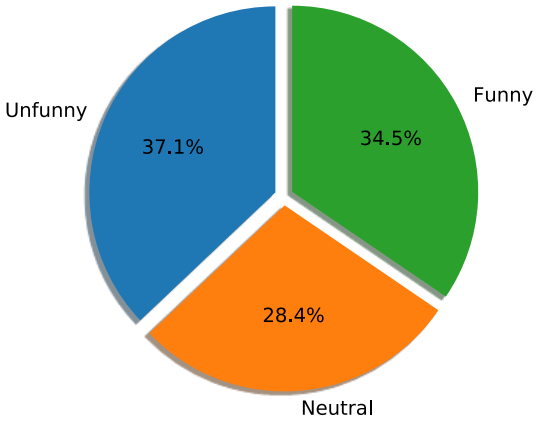


Figure 2, below, validates the manner by which jokes labeled. It shows that the chosen labeling method leads to a very even split between N*eutral, Funny* and U*nfunny* jokes.

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. 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. Further details on this benchmark implementation are provided in *Section 3.2.1 – Benchmark Implementation*.

Decision trees can learn non-linear relationships that linear logistic regression models 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[5]. The random forest model trains several decision tree models independently and averages their results. Bagging of observations and consideration of a reduced of feature set at each ensures that that a random forest model generalize better than a single decision tree.

Neural networks are even more flexible than decision trees. An important advantage of neural networks is that they can account for word order, linear logistic regression and decision trees. The will be discussed more in *Section 3.1.2 – Dense Word Embeddings.* The drawback to neural networks is that they can be slow to train and require the adjustments of many different 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:
    a. Dropping observations with between 2 and 9 upvotes (inclusive)
    b. 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
    a. A train data set with 196,509 observations
    b. 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 sklearn's[6] 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.

[5] L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001.

[6] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

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 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 in *Section 3.2.1 - Benchmark Implementation*.
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 a neural network can 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 i^{th} element of this vector corresponds to the i^{th} 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 these pretrained embeddings incorporates prior knowledge and should help ensure that the neural network models can more easily learn the relationship of interest. I set the length of the sequences to be equal to 30, since 96.7% of the jokes are 300 words or less.

1. The sklearn CountVectorizer class used to create a vocabulary that assigns words to indexes
    a. The NLTK[7] package's *wordpunct_tokenize* function was used in place of sklearn's default tokenizer
2. The embedding matrix was created per the following steps:
    a. The pre-trained GloVe word vectors were loaded from disk into a dictionary
    b. A random matrix was created
        i. Number of rows = length of vocabulary (number of words) + 1
        ii. The first row was set equal to 0 and reserved for the padding values in step 3.c
        iii. Number of columns = dimensions of word vectors
    c. For every word in vocabulary, the corresponding row of the embedding matrix was updated
        i. if there was no corresponding pretrained word vector, the word's entry in the embedding matrix was left in its random state

[7] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

3. Sequences
    a. Sequences were created by splitting sentence into lists of tokens using nltk's *wordpunct_tokenize* function
    b. The corresponding index for each word identified using the vocabulary from step 1.
    c. The list was then transformed into an array, each with a length of 300.
        i. If the sequence was shorter 300 words, it was padded with trailing 0s (that referred to the first row in the embedding matrix)
        ii. If the sequence was longer than 300 words, it was truncated.
4. A validation set was created by splitting the train sequences using an 87.5%/12.5% split.

3.2 – Implementation

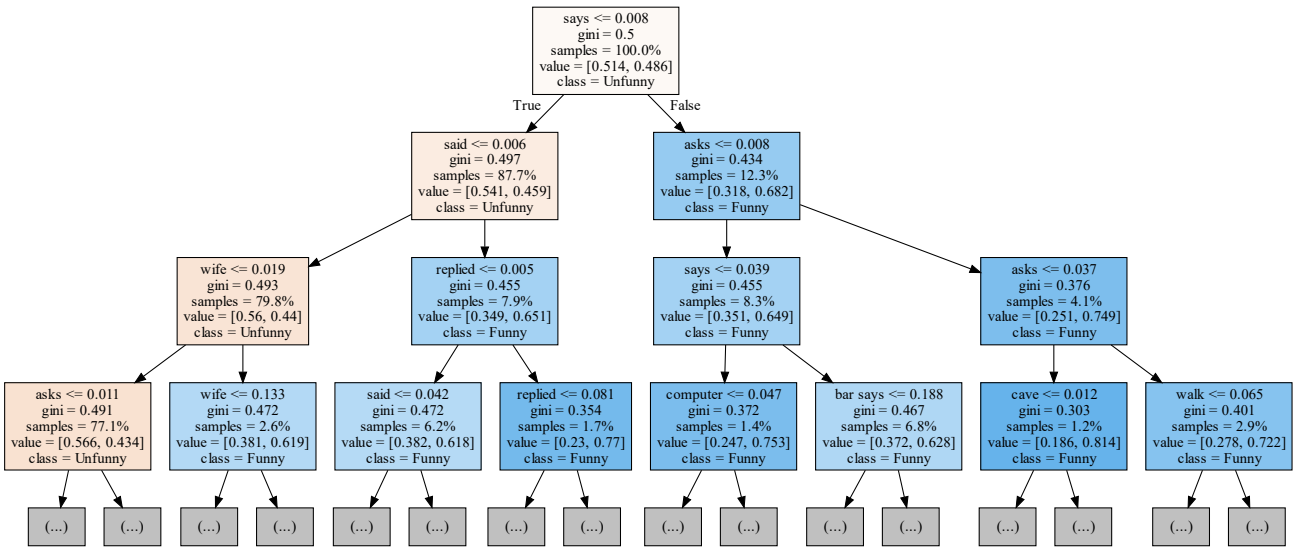
3.2.1 - Benchmark Implementation

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 far from 0. The best C value was found using 8-fold cross validation grid search. I tried 8 values of C, ranging 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 the best results.

In addition to searching for the optimal C value, the development of the benchmark linear model was 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. These values corresponded to 12.5% to 100% of the maximum number of terms in the unrestricted document term matrix. As mentioned in *Section 3.1.1*, it was determined that a *max_features* setting of 231,850 provided the best results. This means the document term matrix did not undergo any additional 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. The following figure shows the first three layer of the initial, untuned, decision tree. Blue boxes indicate nodes contain more funny observations than unfunny. Darker shades indicate purer nodes. The top line of text in the box indicates the splitting criteria.



3.2.3 Neural Networks

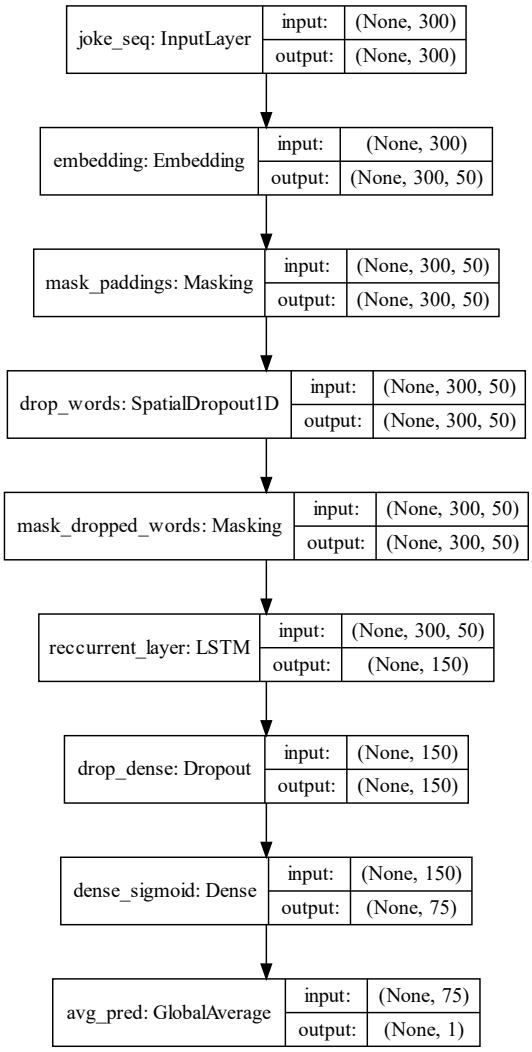
The neural network models were developed using the Keras[8] package frontend to the TensorFlow[9] backend. The Tensorflow computations were executed on my GPU in order to obtain significant speed ups. The main component of the neural network model was a recurrent neural network layer. The recurrent layer processes each word in sequence, by referring to the results of processing the previous word. This means that recurrent neural networks can learn relationships between words based on ther order in which they appear. Figure 3 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 details on each layer of the model architecture. *Section 3.3.3* explains refinements that were made to this architecture. The neural network was trained with the Adam[10] 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.

[8] Chollet, Francois, et al. "Keras." *Github* https://github.com/fchollet/keras (2015).
[9] https://www.tensorflow.org/
[10] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014)

TABLE 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[11] 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[12] (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

FIGURE 3



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:

[11] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research* 15.1 (2014): 1929-1958.

[12] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

Table 4

Parameter	Definition[13]	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

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[14]. 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 that were considered.

Table 5

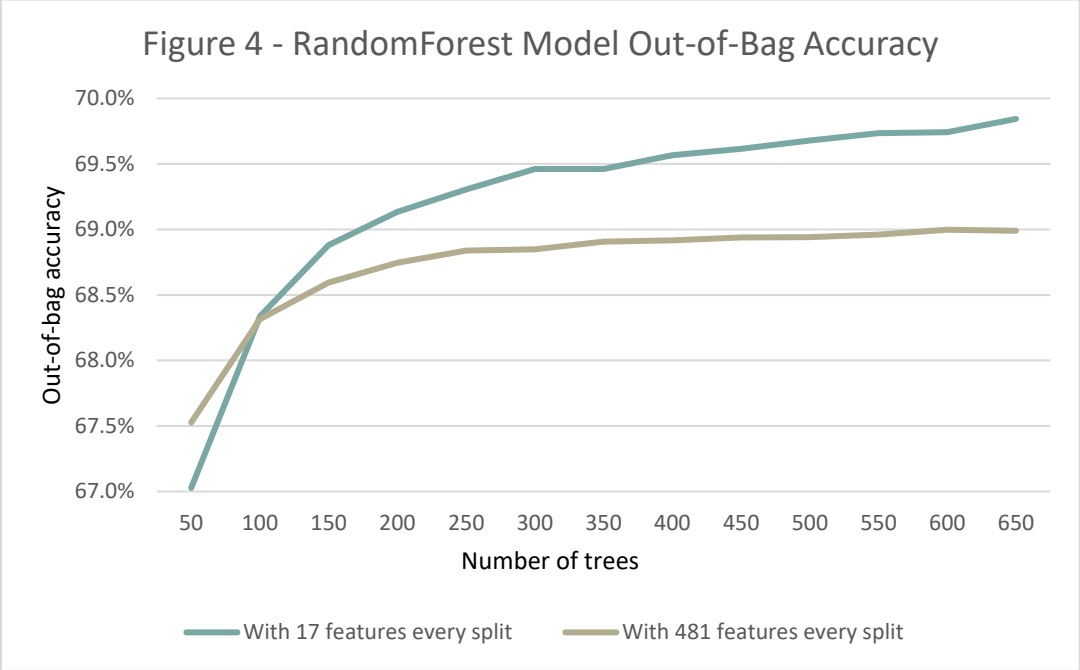
Max_features	Number of features considered per split
Sqrt	sqrt(231850) = 481
Log2	log2(231850) = 17

Figure 4 displays the relationship between out-of-bag accuracy and the number of trees for the two values of *max_features*. The model that considered 481 features at each split ("sqrt") obtained an out-of-bag accuracy of 67.53% with 50 trees compared to an accuracy 67.03% for the model that considered 17 features. These initial models with only 50 trees were the worst performing. As more trees were added, the out-of-bag accuracy increased. With 600 treed, the model with the best out-of-error had the most trees and considered 17 features at each split. It achieved an out-of-bag accuracy of 69.84%. It was therefore tested on the test set. On the test set it achieved an accuracy

[13] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[14] "An important feature of random forest is its use of *out of-bag (OOB)* samples*: For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging* only *those trees corresponding to boot-strap samples in which z_i* did not *appear.*" Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc.

of 69.55% and a AUC of 0.7549. This is a noticeable improvement compared to the benchmark linear model. From Figure 4, it does not seem that the random forest model is overfitting, since it shows that the out-of-bag accuracy continues to increase as more trees are added. However the best random forest model obtained a train accuracy of 98.55%, which seems to indicate there is not much room for improvement.



3.3.3 Neural Network Refinement

In addition to the initial neural network model outlined in *Section 3.2.3*, I trained an additional three models with slight modifications. 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.

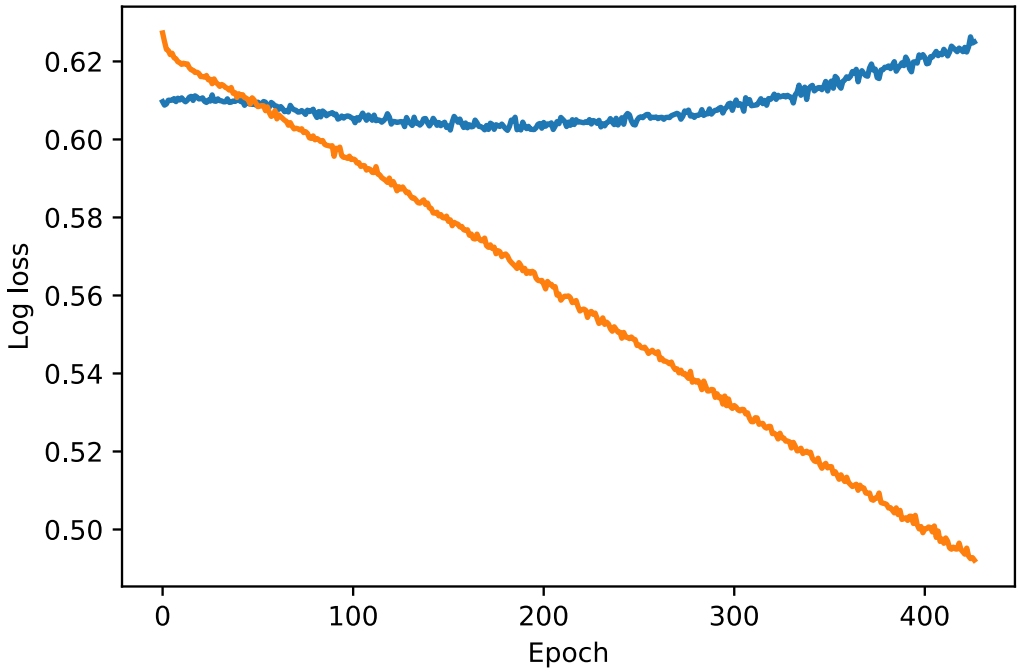
In addition to this modification, I developed two models that were allowed to update the weights of the embedding layer. I did not randomly initialize the weights of these two models. Instead, I used the weights that I had obtained after training the first two models that kept embedding layer fixed. 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 helped prevent overfitting by increasing the dropout rate that was applied to the inputs of the recurrent layer and decreasing the learning rate. Table 6 provides details on the four neural network models that I trained.

Table 6

Embedding Dimension	Trainable Embedding?	Trainable Parameters	Patience	Learning rate	Input Dropout Rate	Best Validation AUC
50	False	131,925	25	0.001	30%	0.7313
300	False	281,925	25	0.001	40%	0.7487
50	True	1,501,725	50	0.00025	45%	0.7609
300	True	8,500,725	50	0.00025	60%	0.7441

As shown in Table 6, the third model with 50 dimensional trainable embeddings obtains the best AUC score on the validation set. It was therefore evaluated on the test set. On the test set it obtains a AUC of 0.7569 and an accuracy of 68.71%. This means that it outperforms the benchmark linear model slightly on both of my preferred metrics. Figure 5 plots train and validation log loss against the epoch number for the best neural network model. The large divergence between validation and train loss is indicative of overfitting, which is surprising given the use of a high dropout and low learning rate. This large divergence should be put into perspective. In the final epoch the average accuracy on the train set was 76.06% and 68.25% on the holdout validation set. This is a much smaller divergence than I observed for the linear and random forest models.

Figure 5 - Neural Network Fit Path



4 - Results

As indicated in *Section 3.3*, the model with the best test AUC was the neural network model, while the random forest model achieved the best accuracy on the test set. If I were to consider these criteria alone, I would choose the random forest model as my final solution. This is because the random forest model is 0.84 percentage points more accurate on the test set than the neural network model, while the neural network model's test AUC is only 0.0020 higher than that of the random forest model. However, there are other factors that should be taken into account. The most important is the fact that the neural network model is much smaller than the random forest model. After being serialized to disk, the neural network model occupies 17 megabytes of space, while the random forest model takes up 8 gigabytes. This means that the random forest model is over 450 times bigger than the neural network mode. This is quite large, and many computers would not be able to load the random forest model into memory. The neural network model is a much more reasonable size. This discrepancy in model size results in a much longer prediction time for the Random Forest model. The random forest model requires 1 minute & 34

seconds to predict the 28,073 test observations while the neural network model takes only 9 seconds. I therefore prefer neural network model and will use it as my final model.

To ensure that the final model can generalize well to unseen data from other domains, I tested it on some data from sources other than reddit. Table 7 shows the predictions that the model produced for some inspiring but unfunny quotes. A higher percentage indicates that the model is more confident that the quote is funny, where a score above 50% could be taken as a prediction that the quote is funny. The model performs quite well on these unfunny quotes, and only predicts that one of them is funny. Table 8 shows some more quotes that I found, but this time they are humorous. The model performs even better on these quotes, assigning each of them a score above 50%. On this small sample the final model obtains 90% accuracy. I am satisfied that the final model is a quite solution to the problem posed in *Section 1.2*.

Table 7 – Unfunny Quotes[15]

Prediction	Text
28.1%	*Find a place inside where there's joy, and the joy will burn out the pain.*
29.7%	*Try to be a rainbow in someone's cloud.*
57.1%	*It is during our darkest moments that we must focus to see the light.*
37.1%	*Keep love in your heart. A life without it is like a sunless garden when the flowers are dead.*
19.5%	*The best and most beautiful things in the world cannot be seen or even touched - they must be felt with the heart.*

Table 8 – Funny Quotes[16]

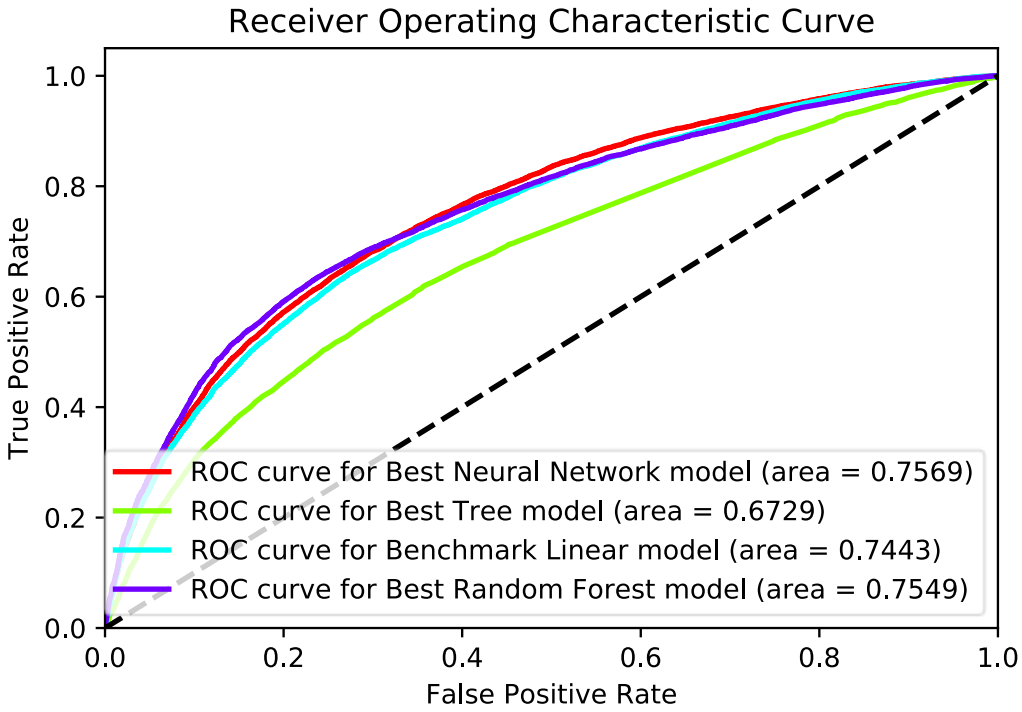
Prediction	Text
76.2%	*If I won the award for laziness, I would send somebody to pick it up for me.*
69.9%	*Me and my bed are perfect for each other, but my alarm clock keeps trying to break us up.*
78.0%	*If we shouldn't eat at night, why is there a light in the fridge?*
76.5%	*Sometimes I wish I was an octopus, so I could slap eight people at once.*
76.5%	*Some people are like clouds. When they go away, it's a brighter day.*

5 - Conclusion

5.1 - Free-Form Visualization

The following chart displays the Receiver Operating Characteristic (ROC) curve for the four types of tuned models that I produced. It is very interesting how the benchmark linear, random forest and final neural network models are all so close. It is very hard to notice the slightly superior performance of the final neural network model. The single decision tree model is the outlier in this chart, which is not surprising given its poor AUC.

[15] Source: https://www.brainyquote.com/top_100_quotes
[16] Source: http://www.coolfunnyquotes.com/category/top-100-funny-quotes/



5.2 - Reflection

This project developed several different models to tackle the problem of classifying reddit posts as either funny or unfunny. This was meant to be a proxy for humor in general, so I avoided incorporating extra metadata such as the name of the author or the date of the submission. The models spanned a range of complexity that went from simple logistic regression to more complex recurrent neural networks. In the end, the neural network model was chosen as the final solution model since it achieved the best AUC on the test set (by a small margin) and was much smaller and faster to predict than the random forest model it. In the end I am satisfied with the results that I obtained, especially for the quotes. I found it surprising that it was so hard to beat the benchmark linear model. In a business setting, it is quite probable that the logistic regression model would be preferred to the neural network model since it was much faster to develop and it is easier to explain. The reason it is so difficult to beat benchmark model is probably due to the fact that there is relatively limited amount of training data. In addition, humour is subjective and reddit upvotes are noisy. This means that even a human would have a difficulty in correctly classifying many of these posts.

5.3 - Improvement

If I were to iterate on this project, I would want to transition to the generation of jokes. This would require the development of another neural network model. Hopefully there is some method to incorporate the neural network classifier that I have adopted as my final solution. The generation of jokes would also require a great deal more training data. This would be necessary to ensure that the joke generator learns grammar can connect ideas together. It would not be necessary for this enhanced data set to contain only jokes. It could contain any type of text that properly structured with coherent grammar and ideas. Once I had developed a neural network model that could generate coherent texts, I would fine tune the model on a data set that only contains jokes or humorous stories.