STOR 565: Sarcasm & Scores in Reddit Comments

Team Name: The Gradient Dissenters

Names: Paul Song, Ethan Wu, Nidhi Murlidhar, Jonah Soberano, Stephanie Jung

Summary:

For our STOR 565 Final Project, we analyzed comments from Reddits and applied machine learning techniques to predict some features. Our dataset, obtained from Kaggle, is made up of around 1 million comments from Reddit, a social news website where users can post and comment on a variety of topics. Each comment was accompanied by information about the comment, such as the parent comment, the subreddit where it was posted, and the date it was posted. We came up with two goals for our analysis of this dataset. The first was to predict whether a comment was sarcastic or not, where a comment is considered sarcastic if it has the sarcasm tag on it ('\s'). The second goal was to predict the score of the comment, where the score of a comment is the sum of the upvotes and downvote the comment received.

To analyze the data, first we had to sanitize our dataset. In doing this, we found that we needed to eliminate the "number of upvotes" and "number of downvotes" columns from our data because they did not make sense. Given that our dataset did not contain many features to predict with, our next step was to create more features from our comments. To do this, we used a MapReduce function, which parsed through each comments and created a table of features for each comment, like the number of exclamation points, or the number of capitalized letters.

After creating these features, we first analyzed our dataset using logistic, LDA, and QDA models. With these initial models, we found pretty high error rates, so we wanted to improve our models. We tried to create more useful features, added additional models, such as random forests neural networks, and incorporated natural language processing techniques. From our results, we found there were significant improvements in our accuracy and error rates when considering the neural networks. Though we did not achieve an excellent model statistically, our work shows that it is indeed feasible to create models that are able to predict sarcasm in comments and the score of the comment. We hope in future endeavors we can try and find other ways to improve our models.

Introduction to the Dataset

For our project, we chose to focus on Reddit comments. Reddit is a very popular social media site where people can post all sorts of links, text, pictures, videos, and etc. and other people can "upvote"/"downvote" or even comment and respond to the post. Users of reddit can join different subreddits, which are dedicated pages where Reddit users can subscribe to and focus on a certain topic. For example, the r/sports subreddit has many posts about news and events in sports all around the world. The exact dataset we used was from kaggle.com and can be found at https://www.kaggle.com/sherinclaudia/sarcastic-comments-on-reddit/kernels**.** We did not use any of the kernels from other users for our analysis. We were interested in working with this dataset because most of our group uses Reddit on a daily basis and we are interested in what makes an online comment sarcastic, or what "types" of comments get higher scores. Specifically, our motivation with these questions is to gain interesting insight on what makes a comment sarcastic or popular on Reddit. From this dataset, we can see we have about one million data points each with ten features. The ten features are a label for sarcasm, the comment, the author of the comment, the subreddit of the comment, the score of the comment, the up arrows of the comment, the down arrows of the comment, the date of the comment, the UTC time of the comment, and the parent comment, which is the comment to which the primary comment replied. Here, the label for sarcasm was based on whether a comment contained "\s," which is how Reddit users conventionally indicate sarcasm in their comments. Our dataset was split about 50/50 in terms of sarcasm and not sarcasm, and the score of the comment is a based on the number of up arrows minus the down arrows.

Goals

Our goal with this dataset is to see if it was feasible to answer two questions through machine learning. The first is whether we are able to correctly predict a comment having the sarcastic tag, while the second is whether we can predict the score of the comment. These two questions are classification and regression problems respectively. Though our objective is clear, the path to get good predictive models is rather difficult. Our group does not have any experience with the standard natural language processing techniques which would have been effective in analyzing our dataset. So, we initially sought to think of our own methods to parse through comments and turn them into useful features for our models to use. This will be further explained in our pre-processing section.That being said, we also wanted to compare our model to a model that actually uses natural language processing.

Pre-Processing

Before any big data analysis could be done, we needed to clean our dataset. There were several blank comments, so we deleted them. We also deleted a comment that spanned over multiple lines in the csv file and did not seem useful at all. We noticed that the "ups" (upvotes) and "downs" (downvotes) features of the original dataset made no sense as some comments had a negative amount of upvotes and downvotes. Consequently, the amount of upvotes minus the amount of downvotes did not match the score of the comment, despite that generally being how Reddit comment scores are calculated. So, we decided to ignore those columns. Finally, we added a column for id numbers for each comment to make referencing them easier in the future.

As mentioned previously, our group initially focused on finding a way to analyze the reddit comments from our dataset into useful features that our machine learning models would be able to use effectively. We decided to compile a list of categories that we thought may indicate sarcasm and that we could get by running a computer program to parse through the comments, such as the use of several exclamation marks in a row, the use of ellipses, or the use of swear words. Later, we also used our program to create features from other categories, like the date of the comment, what subreddit the comment was from, etc.

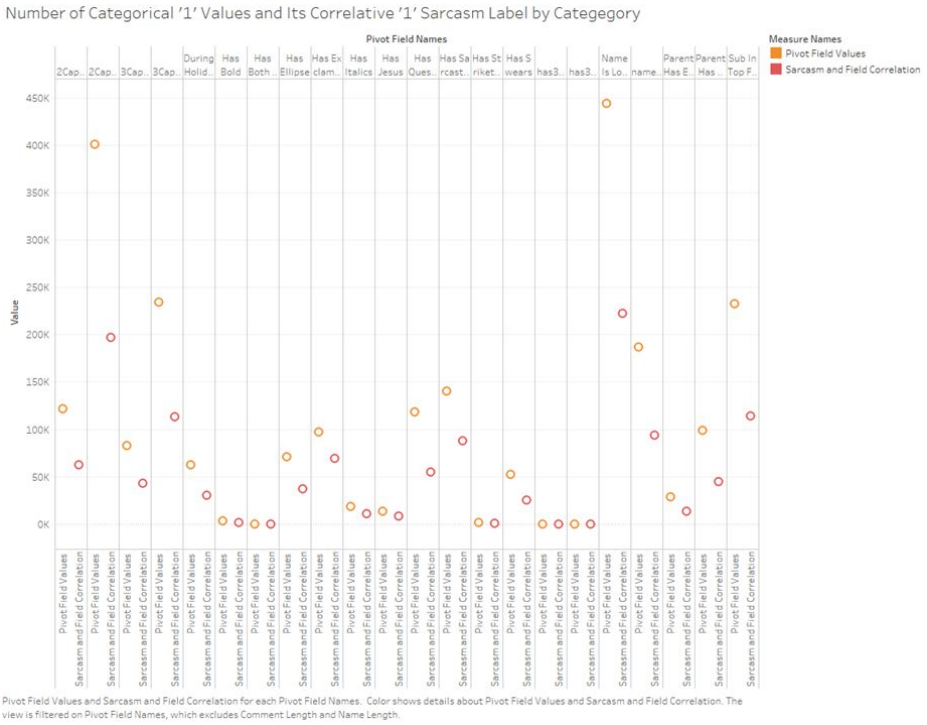
In order to achieve this in a timely manner, we wrote a MapReduce java program to create a new dataset with the features we want. MapReduce is a type of program mainly used on very large datasets that uses the idea of parallelism to execute code efficiently. Initially, we split the dataset and mapped each split in a key value pair. We then combined and partitioned them to reduce them into our desired results. The advantage of this method is that we can analyze our large dataset very quickly and with ease compared to using clunky Excel functions, like COUNTIF, for our dataset that has over one million data points; however, we need a dedicated cluster that does this parallelism for us. One of the members of our group is in a "Big Data" computer science course involving this type of program. The cluster is hosted by the computer science department, and this allows us to use MapReduce. (See Sarcasm.java, SarcasmMapper.java, SarcasmReducer(numCapital).java for a sample MapReduce code used)

After running all of the MapReduce programs, we compiled all our results to create a csv file that contained all the features we wanted to use. Below is a short description of all our features we created for our initial models and brief descriptions for each of them.

- *id* - id numbers for each data point
- *label* - binary label for whether the comment has the sarcastic tag ('1' indicates sarcasm tag, 0 indicates lack of)
- *score* - numeric score of the comment calculated by the number of upvotes minus the number of downvotes

- *2CapitalInARow* - binary label whether the comment has 2 capital letters in a row
- *2CapitalTotal* - binary label whether the comment has at least 2 capital letters total
- *3CapitalInARow* - binary label whether the comment has 3 capital letters in a row
- *3CapitalTotal* - binary label whether the comment has at least 3 capital letters total
- *duringHoliday* - binary label whether the comment was posted during a major holiday
- *hasEllipse* - binary label whether the comment has an ellipse
- *nameHas2Numbers* - binary label whether the username of the commenter has at least 2 numbers
- *has3Exclamations* - binary label whether the comment has at least 3 exclamation marks in total
- *has3Questions* - binary label whether the comment has at least 3 exclamation marks in total
- *hasSarcasticAdverb* - binary label whether the comment has an adverb that may lead to sarcasm (very, obviously, totally, clearly, really, so, oh, great)
- *hasBold* - binary label whether any part of the comment is bolded
- *hasBothMarks* - binary label whether the comment has both an exclamation mark and question mark in a row
- *hasExclamation* - binary label whether the comment has an exclamation mark
- *hasJesus* - binary label whether the comment uses “Jesus” or “Christ” or any other similar words
- *hasItalics* - binary label whether any part of the comment has italics
- *parentHasExclamation* - Binary label whether the parent comment has an exclamation mark
- *parentHasQuestion* - Binary label whether the comment has a question mark
- *hasStrikethrough* - Binary label whether any part of the comment is striked through
- *hasSwears* - Binary label whether the comment contains swears (will not list here)
- *subInTopFifty* - Binary label whether the comment was in the top fifty subreddits
- *nameIsLowerCase* - Binary label whether the username of the commenter was in all lower case

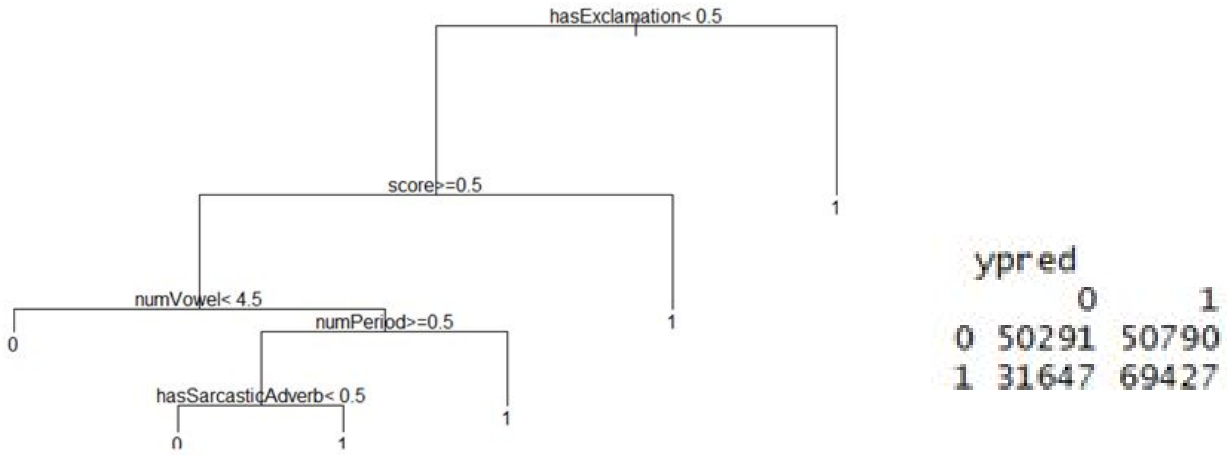
Note that there seem to be some redundant features as we did not know a good threshold to the number of punctuation or etc. that might give some indication for sarcasm. We chose to keep all of these features and see how they do in our initial models.



Above is a brief overview of our data. The left orange dot represents the number of datapoints that has the feature as 1, while the right red dot represents the correlation of that feature. We can see that a lot of our features do not apply to a lot of our data points. However, we hope that a combination of our features will enable our models to still predict sarcasm effectively.

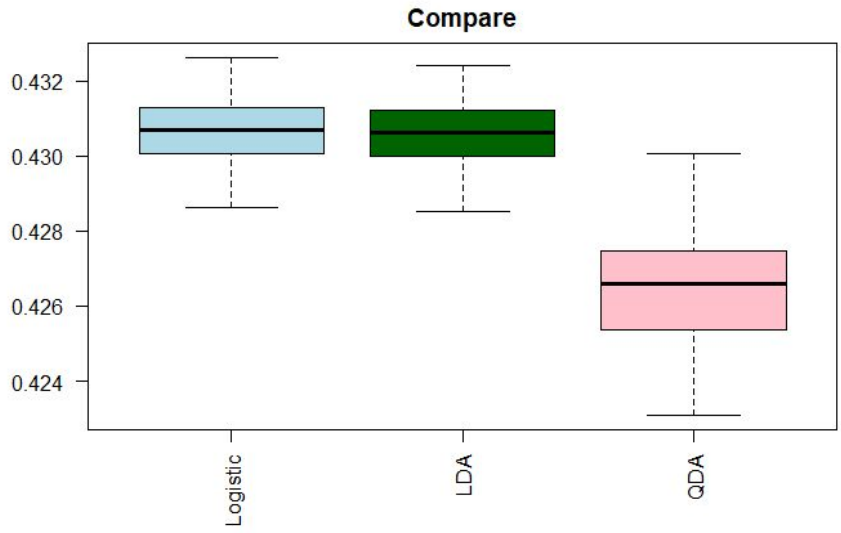
Initial Models

Our initial models solely comprised binary classification techniques categorizing comments as sarcastic/not sarcastic using the binary features from the above table as predictors. We implemented decision trees, LDA, QDA, and logistic regression. Of these methods, the decision tree was the most accurate, with an accuracy rate of 59.228% with an 80/20 training/test split; the resulting tree is shown below. (Sensitivity-.6138, Specificity-.5775)



Our Decision Tree and its Corresponding Confusion Matrix

For LDA, QDA, and logistic regression, we ran each model 100 times with different training/test segmentations of the data (also with an 80/20 training/test split). All three models performed similarly -- though slightly worse than the decision tree -- with average error rates of 43.1%, 42.6%, and 43.1% respectively when all predictors in the dataset were used. (See LDA_QDAInitail.rmd for the code)



Overall, a lot of these models did not perform especially well in classifying comments as sarcastic, likely due to the binary nature of the features in our dataset. Moving forward, we strived to improve the overall

performance of our models both through reworking the data by diversifying our features as well as through incorporating other techniques.

More Data Analysis

Seeing most of our initial models not doing very well in classifying sarcastic comments, we decided to add 3 more approaches. Firstly, we added non-binary features to our dataset. Again, we used a MapReduce program to achieve this; the additional features we made are below.

- *commentLength* - The number of characters in the comment
- *nameLength* - The number of characters in the username of the commenter
- *numAdverb* - The number of adverbs that may indicate sarcasm in the comment
- *numCapital* - The number of capital letters in the comment
- *numComma* - The number of commas in the comment
- *numExclam* - The number of exclamation marks in the comment
- *numJesus* - The number of uses of “Jesus”, “Christ”, or other similar words in the comment
- *numPeriod* - The number of periods in the comment
- *numQuestion* - The number of question marks in the comment
- *numSwear* - The number of swears in the comment
- *numVowel* - The number of vowels in the comment

We decided not only to add non-binary features that we thought would correlate to sarcasm, but also to create some random categories, such as number of vowels, to see if such features have any correlation with whether a comment is sarcastic. Moreover, given these new features, we also wanted to use regression to predict comment scores. This would give us more insight on how to get a higher score comment on Reddit. (see SarcasmData.csv to see the full altered dataset)

The second approach is to see if neural networks can significantly improve our test error for classification. Neural nets have become very popular recently as they are effective with very large amount of data, like our dataset. We hope to see some improvement in classification, but also to see its effectiveness in predicting the score of the comments.

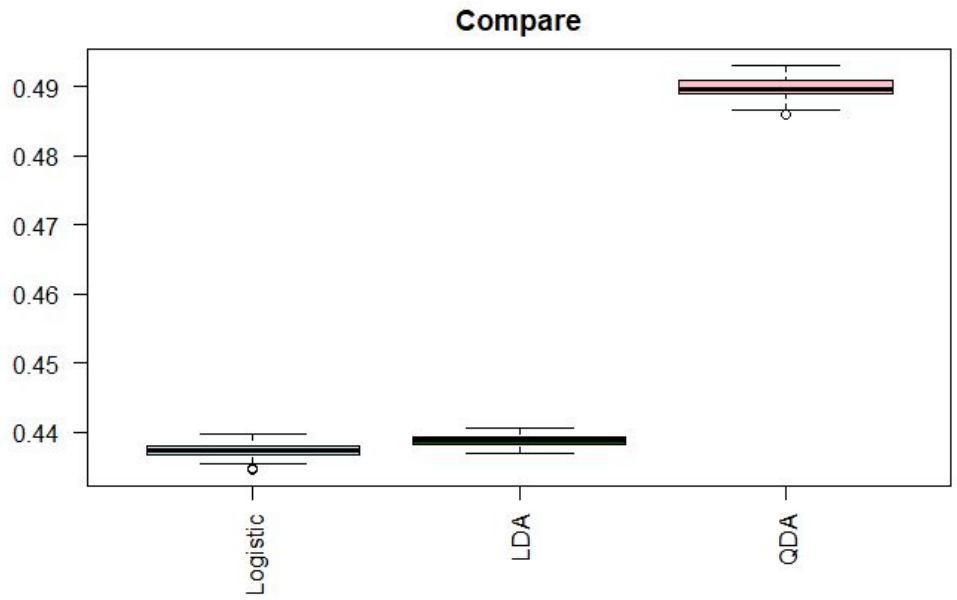
The third new approach is to use a natural language processing model to both classify the comments by sarcasm and predict the score of those comments. This model will be able to “understand” the words used in the comments and should probably be more effective in predicting than our models; consequently, it would be a useful tool for comparison and assessing the overall quality of our other models.

Models and Analysis

Classification - Sarcasm

Logistic/LDA/QDA

To predict whether or not a comment was sarcastic, we decided to implement discriminant analysis to determine an approximation for the Bayes classifier of the sarcastic response variable. We used non-binary features in these models to measure if they were more accurate than our initial models. Our results show that the error rates of the LDA/QDA models without binary features were higher than those of our initial model, at around 43.7%, 43.9%, and 48.9% for the Logistic, LDA, and QDA models respectively. Looking at the confusion matrix, we found that the Logistic and LDA models had similar results with specificity of .298 and a sensitivity of .826, while the QDA had a specificity of .947 and a sensitivity of .0654. With such a high specificity and a low sensitivity the QDA model was essentially predicting most of the comments to be not sarcastic (0), which means that the predictions had a high number of false negatives. Using these results we determined the LDA to be the better discriminant model and the logistic model to be the best model of the three tested. (see LDA_QDA.rmd and LDA_QDA_Logisitic.rmd for the code)



Random Forests

We also performed random forests to see if we could improve our initial decision tree model, with multiple trees. For the decision trees, we attempted to classify both the Sarcasm Label and whether the Score associated with the comment would be positive or negative (indicating a positive response to the

comment). We initially tried to achieve this by predicting the Sarcasm Label; we started with 3 trees, then 10 trees, and 100 trees. To the right is the confusion matrix for the 100 tree model resulting in a 62.8% accuracy, much better than our initial model. However we didn't notice that much improvement between 10 trees and 100 trees, and the runtime difference between the two was too great, so we decided to predict the Score as a binary variable (positive or negative) with only 10 trees and achieved the following results with a 87.39% accuracy. The confusion matrix for this random forest is displayed to the left. We can see from this that it is much harder for models to predict whether a comment is sarcastic, while it is much easier for them to predict positive/negative scores, though the sample size of negative scores were rather small. (See DecisionTree_RandomForest_PositiveScore.rmd and DecisionTree_RandomForest_Sarcasm.rmd for code for both models)

	y_pred	
	0	1
0	66617	34464
1	40555	60519

	y_pred	
	0	1
0	46	25297
1	188	176624

Neural Networks

With the recent popularity of neural networks, we decided to create a neural network with the hope of developing a more improved sarcasm classification model. Neural networks have many parameters to tune, but they take a very long time to create, especially with a very large dataset like ours. Our model was based on one of our group members' previous projects using neural networks. For classification, we decided to use ReLU as our activation function for the hidden layer and a sigmoid activation function for the output layer. We used "categorical_classification" as our loss and "accuracy" as our metric. This is one of the standard ways for doing binary classification using neural networks. We used all of the features and hope that our network can find the useful information to predict sarcasm. We also looked at different activation functions and different number of nodes in the hidden layer. However, since neural networks take a very long time to train, we had to compare models separately using a validation set. We had the following hidden layers with their corresponding accuracy:

- 1 layer 8 relu nodes: 61.62%
- 1 layer 32 relu nodes: 62.52%
- 1 layer 128 relu nodes: 62.11%
- 1 layer 32 tanh nodes: 61.74%
- 1 layer 32 sigmoid nodes: 62.47%
- 2 layer 32 relu nodes: 45.61%

Note that some of the differences between the accuracies are negligible; however, this means we can just pick the "best" one as all the other models would have similar results. We determined the "best" model

would be a network with one hidden layer with 32 ReLU nodes. We reran the model and got a test accuracy of 62.44%. Below is the confusion matrix for the model.

	0	1	
0-	[68355,	32414]	Sensitivity: .611
1-	[43521,	57865]	Specificity: .641

We can see that our neural network does a better job of recognizing both not sarcastic and sarcastic comments. Though the accuracy is a bit low, our models shows that neural networks indeed do a better job at predicting sarcastic comments, and can probably be improved upon to get a even better accuracy. (see NeuralNetSarcasm.py for the code)

Natural Language Processing

For comparison, we also used natural language processing to create a binary classification model with the intention of assessing the overall accuracy of using the predictors we defined earlier vs. using the actual text of the comment itself as a predictor for sarcasm. Since, as mentioned earlier, none of the members of our group are extremely well versed in natural language processing techniques, we followed a tutorial online that used Keras and TensorFlow for binary text classification in R. In keeping with the tutorial, we chose to use ReLU as our activation function for the hidden layer and sigmoid as our activation function for the output layer. We also used binary cross-entropy for our loss, Adam for our optimizer, and accuracy as our metric. The tutorial can be found here:

https://shirinsplayground.netlify.com/2019/01/text_classification_keras_data_prep/

For the most part, we followed the above tutorial; however, we initially split the data into a training and test set (using both 70/30 and 80/20 training/test splits) and used the tutorial to process the text for both the training and test sets. We then fit a model to learn Word2Vec embeddings.With a 70/30 training/test split, the model had a test accuracy 0.699 with a loss of 0.569; however, with an 80/20 training/test split, the test accuracy dropped to 0.499 with a loss of 7.986. The confusion matrix for the 70/30 split is below:

```
$confusion_mat
    y
yhat      0      1
   0 115798  55225
   1  35939  96286

$stats
```

The huge drop in accuracy from the 80/20 training/test split can be assumed to be caused by overfitting, a factor we need to consider for all models. However, using the 70/30 split, we can see that a neural

network can achieve a very good accuracy using natural language processing techniques. (see NLP_NeuralNet.rmd for the code)

Regression - Score

Linear Regression, Ridge Regression and LASSO

As stated earlier, we also wanted to predict the score of the reddit comments. We first used three regression models (linear, ridge, and lasso) to do this. Before fitting the models, we plotted the predictors against the score, and based on that, we decided to normalize all the scores so that the range from 0 to 1. We also did a 10-fold cross validation for the ridge and lasso models to find the optimal lambda value (~.00144, ~3.337e-6 respectively). Our mean squared errors for each model are:

Linear - 1.841349e-5 → .176346

Ridge - 1.841178e-5 → .1763296

Lasso - 1.841356e-5 → .1763467

Note that these error rates are based on the normalized data. From our data, we can see that our ridge regression model did slightly better than the others. We also looked at the mean absolute errors to see how far on average our errors were by. If we convert the error rates to correspond to the actual average squared distances, we obtain 8.942 for linear, 8.920 for ridge, and 8.940 for lasso.

We also decided to look at best subset selection for the linear model. Looking at 50 generated models, we picked the "best" model based on the lowest Cp score. We fitted a model to the corresponding features and with the normalized data, we got a mean squared error of 1.84133e-5, or transformed as .1763442.

(See LinearRegression.rmd for the code)

Neural Networks

Like we did in classification, we used neural networks to predict the score of each Reddit comment. We converted our previous neural network that makes sense in doing regression. We used mean absolute error as our loss and as our metric. We again tested multiple models, however, from experience with regression neural network, our group decided to only check for the number of nodes in the hidden layers as we used the "relu" activation function for the hidden layer and a "linear" output layer. We used all of our features, and here are the results of testing multiple models with a validation set (error in terms of mean absolute error):

- 1 layer 4 relu = 6.735
- 1 layer 8 relu = 6.727

- 1 layer 16 relu = 6.723
- 1 layer 32 relu = 6.722
- 1 layer 64 relu = 6.723
- 2 layer 32 relu = 6.723

Again we see that our "best" model is with one hidden layer of 32 relu nodes. Also note that once again the differences between most of these models are negligible, so any of them will be fine to use. Using our "best" model, we achieved a test error of 6.724. We can see that our neural network, based on mean absolute error, can achieve a significantly better error rates than our standard regression models. (see NeuralNetScore.py for the code)

Natural Language Processing

Here, we used the same code from the tutorial as we did when classifying comments as sarcastic/not sarcastic, with minor modifications -- we changed the loss from binary cross-entropy to mean-squared error and the metric from accuracy to mean absolute error. We used both 70/30 and 80/20 training/test splits. Using a 70/30 split, we achieved a mean absolute error of 6.932 on the test set, while we achieved a mean absolute error of 6.881 on the test set when using an 80/20 split. Again, we see that our neural network does a lot better than our regression models, however, we do get a slightly worse error rates than our original neural network. This may indicate that the score of the comment may be less dependent on the actual content of the comment, but other factors (i.e. which subreddit the comment was in/what time the comment was posted), but more testing would be needed to confirm this. (see NLP_NeuralNet.rmd for the code)

Conclusion/Future Endeavors

One model we attempted but could not obtain meaningful results from was the SVM model. Using standard SVM procedures we tried to classify our sarcasm data but found that it took too long to process. Some further research yielded that SVM with around one million rows took around 24 hours to run. Since the time to find the optimal parameter for the SVM would take even longer, we did not attempt to use the entire dataset. Our next approach was to take subsets of the dataset to see if we could run SVM at a more efficient time, but the most practical size was around ~1000 rows, which we decided was too small of a subset to model with in any meaningful way.

Another model we unsuccessfully attempted was k-nearest neighbors. As in our homework, we used Euclidean distance to determine nearest neighbors. Given the nature of our initial dataset (almost exclusively binary predictors), this naturally resulted in too many ties that the model had no way of

breaking. In the future, we might attempt to use k-nearest neighbors with a different distance metric that is more suited to binary predictors.

Due to time constraints, we were not able to extensively test for feature selections, PCA, and additional parameters for neural networks (such as deeper nets, different ways to scale our data). Additional things we could have looked at were sentiment analysis and more detailed analysis on the parent comments. Sentiment analysis would allow us to see what tone a comment conveys, which may even correlate to sarcasm. Sarcasm is also more easily identified with context of the comment, which we have in the parent comment. If we had done more analysis on the parent comment compared to the main comment, we may have been able to observe a shift in tone/meaning from our natural language processing or sentiment analysis.

The idea of using machine learning to predict a quality as nuanced as sarcasm is not just interesting. It also has practical applications in artificial intelligence/speech recognition technologies and could be incorporated into sentiment analysis which could be used for opinion mining, a field that is extremely useful to tech companies in the social media/advertising space. One product idea we had that could easily implement the classifier we built would be a sarcasm detector where someone could type in their own comment and see how sarcastic it was. Similarly, we could implement a popularity detector that predicts the score a potential comment would receive.

Overall, our main goal of this project was to see the feasibility of predicting sarcasm and the score of reddit comments. All of our work indicates that this is indeed possible, especially with the inclusion of neural networks and natural language processing. We hope to see future work incorporate a more detailed analysis of our data and models and achieve a significantly higher accuracy in predicting sarcasm.