Team 18: Predicting Reddit Scores with ML

John Fang
johnfang@andrew.cmu.edu
Carnegie Mellon University

Dongyu Li
dongyul@andrew.cmu.edu
Carnegie Mellon University

Ziheng Cai
zcai@andrew.cmu.edu
Carnegie Mellon University

Zhou Yu
zhouy2@andrew.cmu.edu
Carnegie Mellon University

Yiwen Xu
yiwenxu@andrew.cmu.edu
Carnegie Mellon University

Tingyu Lan
tingyula@andrew.cmu.edu
Carnegie Mellon University

1 Introduction

1.1 Problem Statement and Motivation

In this project, our goal was to build a pipeline for evaluating different machine learning tasks on the task of predicting Reddit comment scores given information about the comment and possibly other information present at the time of posting. Reddit is an online forum where users can submit content and comment on others' submissions. Users can also up-vote or down-vote comments, and a comment's score is a function of the number of up/down-votes that it received. We believe that predicting how well a comment is received when it is posted can have useful applications if done accurately enough. For one, since online forums are usually moderated by volunteers that use their own time to review comments, we believe that this information could be utilized to help direct moderator's efforts to potentially problematic comments and streamline their job, thus helping them save time and effort. Another possible use case is for ranking systems, where comments that are expected to have better scores may be shown higher up in a comment thread. The "sort by best" option in Reddit is based on this idea.

Some questions we would like to answer in this project and the reasons why we think they are important are as follows:

- is it possible to accurately predict the score of a Reddit comment? The use cases we mentioned are only feasible if the models are accurate and reliable.
- which machine learning technique performs best on this task in terms of accuracy? We would like to explore the best-performing model and determine if it is suitable for our use cases
- what features are most important for predicting score? Feature significance can help us better understand the most influential factors affecting Reddit scores, and help decision makers devote more resources into discovering and monitoring the key features of comments.
- how expensive is it to use each model? While machine learning researchers aim to achieve better accuracy using sophisticated models, from the perspective of a company decision maker, the affordability of the model is also a key factor. We would like to determine if it is feasible to train and deploy an inexpensive model with high accuracy. In other words, we want to find a model on the Pareto frontier.

1.2 Dataset

We use the Reddit Comment Corpus, an online dataset that contains data and metadata about comments scraped from December 2005 to October 2019 in JSON format and stored as compressed files. We used the data from October 2017, as that month was one of the largest files provided in bzip format, which was convenient for reading directly into Spark. That month's file was 9.96GB when compressed and 56.42GB when decompressed. After processing and subsampling, which is discussed in Section 3, we ended up with a dataset had approximately 31M data points.

An example of one data entry from this dataset is shown below:

```
{"author":"srvwd", "author_flair_css_class":"", "
    author_flair_text":"I need Pinot Grigio at all times", "
    body":"Get Lala over to the housewives of ATL.", "can_gild
    ":true, "controversiality":0, "created_utc":1509494400, "
    distinguished":null, "edited":false, "gilded":0, "id":"
    dp61rmc", "is_submitter":false, "link_id":"t3_79vht1", "
    parent_id":"t3_79vht1", "permalink":"/r/BravoRealHousewives
    /comments/79vht1/
    i_will_continue_to_post_cryptic_pics_of_my_sugar/dp61rmc/",
     "retrieved_on":1511998504, "score":3, "stickied":false, "
    subreddit":"BravoRealHousewives", "subreddit_id":"t5_2v6dk
    ", "subreddit_type":"public"}
```

2 ML Methods and Metrics

2.1 Methods

We selected three machine learning techniques that can be applied to regression problems: linear regression, random forests and multi-layer perceptrons. All three methods were trained to minimize the mean squared error loss on a training split representing 80% of the full dataset. We used a validation split for tuning hyperparameters and test split for evaluation, each of which was 10% of the full dataset. These techniques were compared against a baseline of predicting the average score in the training dataset.

2.1.1 Linear Regression (LR)

The first technique that we looked at was linear regression using least squares. We added a lasso term to the objective, which performs both regularization and variable selection. Variable selection can be useful for determining which features are and aren't useful for predicting the comment's score. We tested coefficients λ in the set $\{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10\}$. We found that the best performing coefficient was $\lambda = 0$, which means the method is essentially just normal linear regression.

2.1.2 Random Forests (RF)

The next technique we looked at was random forests [1], an ensemble technique that trains several decision trees on different subsets of the feature set. The hyperparameters for random forests were the number of forests and the depth of the trees within the network. We tested numbers of trees in the set $\{10, 20, 30\}$ and tree depths in $\{5, 10, 15\}$. Unsurprisingly, the best parameters were 30 trees and depth 15, which were the large values tested for both hyperparameters.

2.1.3 Multi-Layer Perceptrons (MLP)

Finally, we also looked at multi-layer perceptrons. The training step for this method ended up being very expensive in terms of time, so training multiple times to tune hyperparameters was impractical. However, we did use a smaller dataset from 2008 to ensure that the optimizer chosen is able to correctly perform gradient descent with the initial set of hyperparameters we use in training. Our network had two hidden layers with 128 nodes per layer. The optimization was controlled by an Adam optimizer, but initialized with learning rate of 10^{-4} and decay of 10^{-7}. We used a batch size of 32 and trained for 30 epochs.

2.2 Metrics

We selected three metrics for evaluating the accuracy of the ML models and two metrics for evaluating how expensive they would be to use in practice.

2.2.1 Root Mean Squared Error (RMSE)

RMSE is a commonly used metric in regression analysis to measure residuals for in-sample data and prediction errors for out-of-sample data. It is the square root of the second sample moment between the true label values and the predicted values by the models. Although this metric depends on the scale of the data, it still allows us to compare prediction errors of different models for the same dataset and hence it is appropriate to use in our setting. For a model $\hat{\theta}$, the prediction RMSE is

$$RMSE(\hat{\theta}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{\theta}(X_i) - Y_i)^2}$$

where $\{(X_i, Y_i)\}_{i=1}^{N}$ is the test set that has N datapoints. Therefore, its value lies in $[0, \infty)$ with smaller values being better. The square root is for aligning the scale of this metric to be the same as the dependent variable's standard deviation.

2.2.2 Mean Absolute Error (MAE)

In RMSE, the effect of each error is proportional to the squared error between the observed value and predicted value, so RMSE is sensitive to outliers. Thus, we decided to include MAE as another metric to better measure the accuracies of our models without magnifying the effects of outliers, as it measures the average absolute difference between the predicted and true values. The prediction MAE of a model $\hat{\theta}$ for test set $\{(X_i, Y_i)\}_{i=1}^{N}$ is defined as

$$MAE(\hat{\theta}) = \frac{1}{N}\sum_{i=1}^{N}|\hat{\theta}(X_i) - Y_i|$$

Similarly, its value lies in $[0, \infty)$ with smaller values being better.

2.2.3 Coefficient of Determination (R^2)

The coefficient of determination is a metric that can be interpreted as the amount of variance that the model is able to explain. The R^2 metric is usually in the range $[0, 1]$, where a higher number is better. A model that perfectly predicts the results would achieve $R^2 = 1$, while a baseline model of predicting the average label would achieve $R^2 = 0$. A model with worse predictions than the baseline can have a negative R^2 value.

Denote $\overline{Y}$ as the average label. Then we define the following terms for an estimator $\hat{\theta}$.

$$SS_{tot} = \sum_{i=1}^{N}(Y_i - \overline{Y})^2$$

$$SS_{res} = \sum_{i=1}^{N}(Y_i - \hat{\theta}(X_i))^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

2.2.4 Model Size

The model size describes the number of parameters of a model and the required storage space to save the model. Smaller model size makes it easier to distribute the model copies across multiple machines. It also determines how much memory is needed on both training and deployment systems. This metric can be theoretically computed by counting the number of model parameters and empirically measured by simply inspecting the storage spaces the saved models occupy.

2.2.5 Inference Time

The inference time is the time measured for making prediction on a single data point. This metric can be computed theoretically in terms of the number of FLOPs as well as measured experimentally in terms of seconds. Short inference time is preferred over long inference time. Inference time is critical in the Reddit setting, as a plethora of new comments are posted every second, and inference latency in a real-time setting directly affects the throughput and energy consumption of any system that the model is deployed on.

2.3 Model / Technique Justification

The models we selected have several desirable properties that makes them suitable for accomplishing our task and answering the questions we proposed.

Firstly, all of these techniques can be parallelized and thus are suited for handling large datasets. Linear regression can be optimized by splitting the data points and using gradient descent, as covered in class. Random forests is an ensembling technique, so each decision tree can be trained in parallel. Finally, deep learning frameworks such as TensorFlow can handle distributed training automatically.

Secondly, these models are widely used in the industry for various reasons. Linear regression models are often favored for their simplicity and ease of interpretation. We think this might also provide insight on the correlation between those features and the comment score by inspecting the feature importance of a trained model. On the other hand, random forests and neural networks have gained attention due to their high expressiveness and state-of-the-art performance in many tasks [3, 2, 5, 7]. We think these two models provide enough model space to explore whether it is possible to accurately predict the score of a Reddit comment. Although these two models are not as interpretable as linear regression, random forests are still able to provide the importance of features that it looks at.

Finally, these models are relatively small to store, and can be more easily deployed than modern deep neural networks that have parameter counts that are several magnitudes of order larger. Thus, we think that they would be suitable for our use case.

3 Computation

3.1 Pipeline Overview

Our pipeline is illustrated in Figure 1 and is as follows: we downloaded the compressed bzip2 file directly to Azure Blob storage, did our data pre-processing in Spark on Azure Databricks, then wrote the processed data back to Azure Blob. To transfer the raw data to Blob, we downloaded the file to an Azure VM and then uploaded the data from the VM to Blob. For our first two machine learning methods, linear regression and random forests, we continued using Spark / Azure Databricks. For the multi-layer perceptron technique, we copied the processed data to Google Storage and used Google Colab backed by a virtual machine from Google Cloud Platform to train multi-layer perceptrons. We initially opted to use Microsoft Azure because of its support for Databricks, which we were familiar with from the assignments in this class. We decided to write the multi-layer perceptron code in Google Colab, as this seemed to be the most convenient platform that provided an online notebook that we could collaboratively edit and connect to a GPU.

3.2 Machine Specs and Budgeting

All of our code was written in Python 3. The main libraries we used were Spark 2.4, TensorFlow 2.2, and numpy. For the Azure parts of our workflow, we used a cluster with one driver node and up to 5 worker nodes in the "East US" zone. Each machine was a Standard DS4 v2, which has 8 cores, 28 GB and 1.5 DBUs. For the Google parts of our workflow, we used a n1-standard-16 machine in the us-east1-d zone, which has 16 vCPUs and 60 GB memory. This machine was equipped with one NVIDIA Tesla T4 GPU. Due to quota rules on Google Cloud Platform, we were only able to request one GPU and thus could not distribute the computation. However, the TensorFlow Datasets class is able to iterate through a large amount of data without loading everything into memory at once and we can also save model checkpoints; hence, we believe that even though we were not able to distribute the training, our method is still designed to handle scaling to very large datasets without failure.

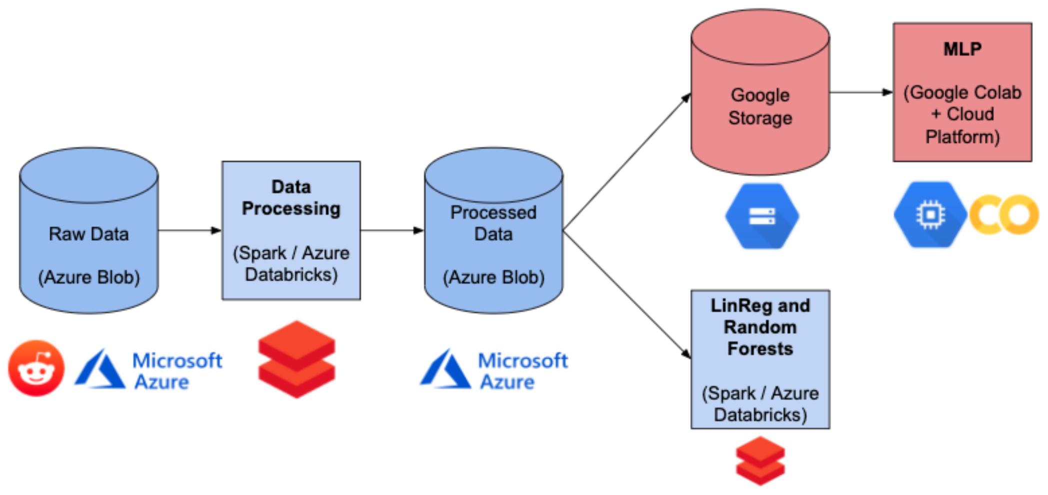

Raw Data
(Azure Blob)
Microsoft Azure
Data Processing
(Spark / Azure Databricks)
Processed Data
(Azure Blob)
Microsoft Azure
Google Storage
MLP
(Google Colab + Cloud Platform)
LinReg and Random Forests
(Spark / Azure Databricks)

Figure 1: An overview of our machine learning pipeline. Steps in blue were done using Microsoft Azure while steps in red were done using Google Cloud Platform.

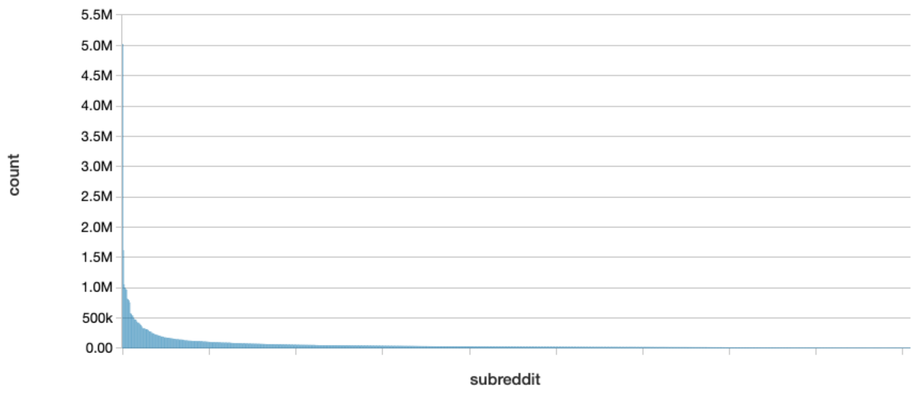


Figure 2: Histogram of the comment count for the top 1,000 subreddits in our dataset

By the end of this project, we spent \$482 on Microsoft Azure and \$41.19 on Google Cloud Platform. For training time (including cross validation), linear regression took 7 minutes, random forests took 34 minutes, and multilayer perceptrons took about 6.5 hours. Preprocessing the whole dataset took about 18 minutes.

3.3 Data Cleaning and Transformations

From the raw JSON files, we extracted several features that we believed may have some correlation with the comment score. For a full list of our features and a description of what they represent, refer to Table 1. The first four features are taken directly from the JSON and represent booleans and categorical features. The categorical subreddit feature only considered the top 100 most frequent subreddits, as there are a large number of total subreddits (over 110K) and the distribution of comments in subreddit has a long tail, as shown in Figure 2. The top 100 subreddits account for about $\sim 41\%$ of all comments. The GloVe embedding is a text embedding for the comment body. The last four features are features that we engineered using information from the dataset but was not immediately available in the JSON fields, such as information from the parent comment. The use of another datapoint's information is justified in our case because the parent comment is available when the child comment is posted. When the model is deployed and a new comment appears, the inference process can access the parent comment's information and feed that data into the model. We ended up with feature vectors of length 157, and the processed data is stored as a CSV file.

Name	Dimensions	Description
controversiality	1	a boolean for if the comment was controversial
edited	1	a boolean for if the author edited the comment after posting
gilded	1	a boolean for if anyone has given Reddit Gold (a paid award) to this comment
subreddit	100	a one-hot encoding for if this comment belonged to one of the 100 most frequent subreddits
GloVe embedding	50	the average word embedding of all words in the comment body using GloVe
parent score	1	the score for the parent comment
parent time delta	1	the time difference between the parent comment and this comment
parent cosine similarity	1	the cosine similarity between the parent's GloVe vector and this comment's
number of words	1	how many words were in the comment body

Table 1: Table of features used in our feature vectors

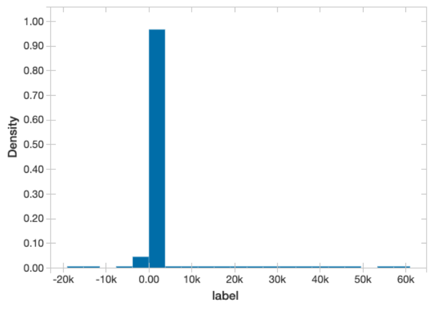
Interesting computations:

1. the features involving the comment's parents are not immediately available in the JSON objects. In order to get this information, we performed a left join of the DataFrame on itself, matching the comment's `parent_id` field against its `id` field. This operation is more advanced than the DataFrame methods we've encountered in this class's assignments and required some troubleshooting, as the non-descriptive feature names and similarity of feature values made it unclear which fields needed to be matched.
2. we initially tested our processing code against a smaller comment set from 2008 and relied on a feature called `name` for the joining operation. However, we later found that the JSON schema is inconsistent across the Reddit Comment Corpus and the October 2017 data did not have the `name` field. After digging around, we found another field `id`, which could be concatenated with a prefix string found in the Reddit API documentation to reconstruct the data we expected to appear in the `name` field. Although this computation ended up being a simple fix from a coding standpoint, it highlights the importance of having some domain knowledge, as we may not have been able to find out how to reconstruct the needed information otherwise.
3. we pre-processed the text body by filtering out symbols, changing the text to lower case, removing stop words, then finally embedding words into vectors of numbers using GloVe. Global Vectors for Word Representation (GloVe) [6] is a pre-trained word representation that maps words into a meaningful high dimensional feature space. After transforming all words to this meaningful dimension, we take mean over all words in a body. Compared to truncating body to fixed word length, this approach keeps the influence of all words while still producing a feature vector with a constant number of dimensions. [4]

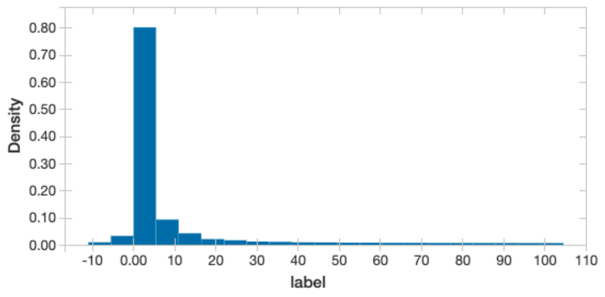
Initially, we did not include any information from parent comment as a feature, and we encountered poor performance resulting in linear regression learning the exact same model as the baseline. After redesigning our process to add engineered features, perform feature normalization, and remove outliers, we observed significant accuracy improvement. From this, we learned the importance of these engineered features, as discussed in the result section below.

3.4 Normalization and Outlier Removal

After processing the data to extract the desired features, the final step of our data cleaning pipeline was removing outliers and normalizing "large-valued" features, which required computing some statistics on the features. We first filtered out comments that had scores within the top or bottom percentile. Figure 3 shows how the range of scores shrank from about 80,000 to a more reasonable range (about 120) after this operation. After this, we subtracted the mean and divided by the standard deviation for the parent score, parent time delta, and number of words features.



(a)



(b)

Figure 3: **(a)** label distribution before removing outliers **(b)** label distribution after removing outliers

	Baseline	LR	RF	MLP
RMSE	8.67	8.2	**7.86**	8.03
MAE	4.35	3.97	**3.63**	4.37
R^2	0	0.1	**0.18**	0.14
Model Size in # Parameters	–	**158**	$\sim$ 4M	$\sim$37K
Model Size in Storage Space	–	$\sim$ **3.3KB**	$\sim$ 42MB	$\sim$ 476KB
Inference Time in FLOPs	–	$\sim$ **300**	$\sim$ 450	$\sim 70,000$
Inference Time in Seconds	–	**0.014 $\pm$ 0.004**	17.01 $\pm$ 0.46	23.38 $\pm$ 0.14

Table 2: Performance of each machine learning technique across multiple metrics. The best value for each metric is highlighted in bold. The “Inference Time in Seconds” result shows the average and standard deviation

4 Results

4.1 Regression Performance

We first address the questions “is it possible to accurately predict the score of a Reddit comment?” and “which machine learning technique performs best on this task?”. We present the metrics of each

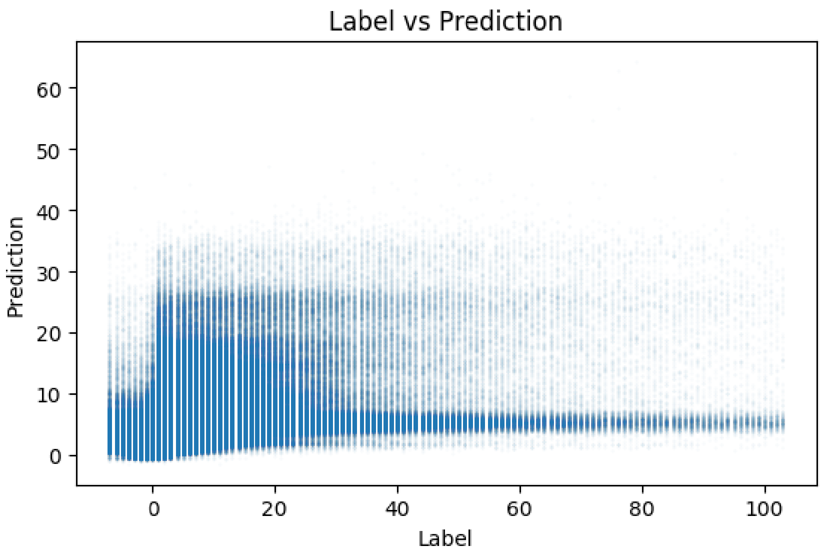


Figure 4: Scatterplot of label vs. prediction for random forests. To better show which areas that are more common, the points have been made partially transparent.

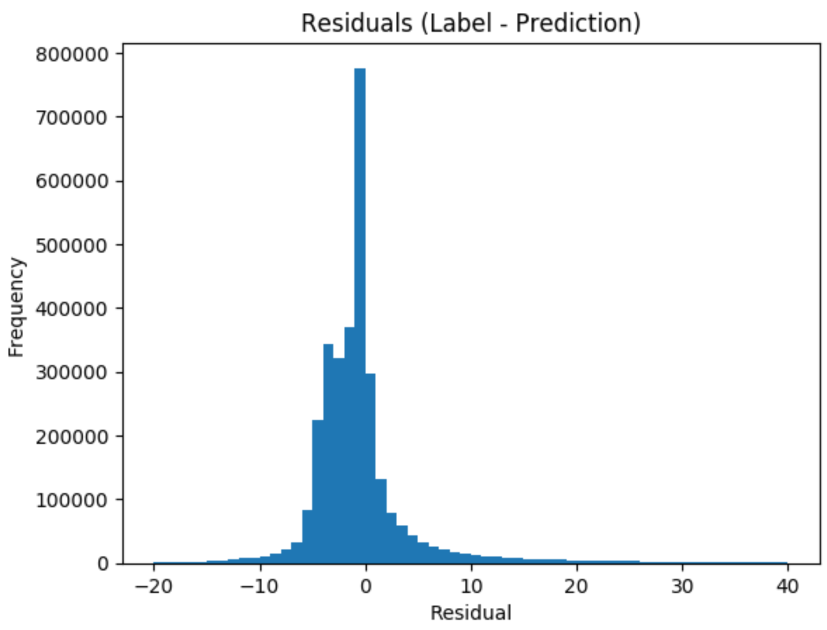


Figure 5: Histogram of Residuals (Label - Prediction) for Random Forests

model in Table 2. To answer this question, we look at the MAE which shows how many points off our predicted scores were. Our results are that the models are around or less than 4 points off on average, which is quite close to the true values and suggests that it is possible to predict scores with fairly high accuracy. In all three metrics that measure the model's accuracy on the regression task (RMSE, MAE, R^2), we can see that random forests dominates and achieves the lowest MAE of 3.67.

We take further analyze at the predictions of our best model (random forests) by creating two visualizations: 1) plotting its predictions against the labels as a scatterplot in Figure 4; 2) creating a histogram of residuals in Figure 5. In Figure 4, we can observe that the model predicts lower values for comments with negative labels, which suggests that the model is able to detect bad comments to some extent, although the predictions themselves are rarely negative. We also observe a tapering effect on the right side, which suggests that the model is more confident in making larger predictions when the comment's label is close to or within $[0, 25]$, but defaults to predicting a value close to the baseline for comments with large labels. In Figure 5, we can see that the majority of residuals are negative, which shows that this model tends to over-predict more often than under-predicting.

4.2 Model Cost

Next, we look at the question "how expensive is it to use each model?" by comparing the model sizes and estimating the parameter counts. Both results are listed in Table 2. Of the three methods, random forests are by far the largest at $\sim$ 42MB, followed by multilayer perceptrons and then linear regression. These results are not that surprising, as a random forest has to store 30 binary decision trees, each of which has up to 15 depth, which means the number of nodes is upper bounded by $30 \cdot (2^{15+1} - 1) \approx$ 2M. Each node must store the feature and the value to compare against. On the other hand, multi-layer perceptrons has to store $158 \cdot 128 + 129 \cdot 128 + 129 \cdot 1 \approx$ 37K numbers for weights and biases and linear regression only has to store 158 numbers for the coefficients and intercept. As one might expect, there is a tradeoff where model with more expressiveness are able to achieve better performance at the regression task but are also more expensive to store. The models we produced are quite small relative to the amount of data that forums must process and store, so we believe they are practical for the described use case. If one needs further control over the size and expressiveness of a model, they have the option to fine-tune the number and depth of trees in the random forest for their specific use case.

In addition to measuring the size of the model, we can also compare the inference time. We compared the models both theoretically in FLOPs and empirically in runtime. For our purposes, we consider

Feature Name	**Importance**
Parent score	0.5478942470329803
Parent time delta	0.21023239114578351
Controversiality	0.048346069617015354
GloVe Dimension 37	0.017113927110949303
Number words	0.016823912913421977
Parent cosine similarity	0.016489298731171786

Table 3: Feature importance according to the random forest model

an addition, multiply or comparison as a FLOP. These results are also in Table 2. Linear regression has the least amount of FLOPs, which naturally follows from the fact that it has the least amount of parameters. Notably, despite having the most amount of parameters to store, random forest has comparable amount of FLOPs with linear regression since it applies branching on the features, making the FLOP count grow logarithmically with the number of parameters. Multi-layer perceptions has the highest number of FLOPs by far, which is not surprising given its large parameter count and the fact that all parameters are used in the inference. An empirical comparison of these models is inherently difficult, as multi-layer perceptrons greatly benefit from the use of GPUs. We compared all trained models on Azure Databricks equipped with only one DS4 v2 worker. We measured the time it took to run inference on the full test split, which represents about 3.1M points, and report the average and standard deviations across 5 runs. As expected, linear regression is by far the fastest method. Random forests and multi-layer perceptrons were both more expensive, with random forests being slightly faster. The discrepancy between the theoretical and empirical results may be due to the fact that decision trees have a lot of branching logic run sequentially. The performance of code with conditionals can be much worse than code without, as incorrectly predicting branches can greatly hinder the pipelining of CPU instructions. Meanwhile, the major computation in linear regression and multi-layer perceptrons is matrix multiplication, which has better spatial and temporal locality and therefore is easier for CPU to optimize. Overall, each of these methods only takes a small fraction of a second to evaluate an individual comment, so we conclude that these methods are feasible for our described use case.

4.3 Feature Importance

To address the question of "what features are most important for predicting score", we can utilize the random forest model that we trained. It is possible to compute the importance of each feature according to the random forest model as a percentage between 0 and 1. We show the top six features and their corresponding importance values in Table 3.

We can make a few observations from this table. Firstly, the top two features, parent score and parent time delta, account for almost 76% of the model's decision, and the importance of subsequent features is small in comparison. These results are not surprising, as one might expect that a comment and a reply posted shortly after would be seen by approximately the same number of people and thus have close scores if they expressed similar ideas. This suggests that context is quite important when evaluating a comment, and we may be able to perform better using information from other comments in the same thread as well as the submission that this comment is associated with.

Another point worth noting is that all engineered features (besides GloVe) showed up in the top six features, which suggests that feature engineering is quite important for doing well in this task. Regarding the non-engineered boolean features, "controversiality" and "edited" are the third and ninth most important features, respectively. On the other hand, the "gilded" feature was ranked quite low and only had 0.09% importance, which seems quite contrary to what one might presume. The Reddit Gold award is often given to well-received comments, so we initially expected to observe a positive correlation between being gilded and the comment score. This may be due to the rareness of gilded comments, as there were only $2,289$ such comments in our training split.

5 Future Steps and Improvement

As mentioned in Section 4.3, it seems from our analysis that engineered features that provide context were the most important. One can imagine there are many other features that we could have incorporated, such as looking at ancestor comments, sibling comments, and the submission that the comment is associated with. The first two approaches are theoretically possible with our dataset, although joining the DataFrame on itself is an expensive operation with $O(n^2)$ time complexity so we only performed one in this project. The last approach would require downloading another dataset (the Reddit Submission Corpus) and also joining it with the Reddit Comment Corpus.

Another interesting direction would be to take advantage of context and the natural tree-structure of Reddit comments in our modeling choices. Since a comment's content is likely related to the history of ancestor comments that preceded it, looking at all comments along the path of the original post to the current comment makes sense. From a modeling perspective, one could explore the use of recurrent neural networks to better model the time sequenced comments leading to any datapoint. Such an approach would be very expensive computationally, because it requires reconstructing the tree structure from the individual raw datapoints.

Besides looking at the context features, we might also be able to improve our model by using a different type of text embedding. In this project, we used an averaging approach to handle variable-length comments. Although one of the highlights of GloVe is that the embeddings have a linear substructure, one could imagine that semantic meaning may be lost when averaging a large number of vectors. An improvement on this would be to explore using models that are designed to embed entire sentences or documents.

References

[1] Jianguo Chen et al. "A parallel random forest algorithm for big data in a spark cloud computing environment". In: *IEEE Transactions on Parallel and Distributed Systems* 28.4 (2016), pp. 919–933.

[2] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[4] Thomas K Landauer, Peter W Foltz, and Darrell Laham. "An introduction to latent semantic analysis". In: *Discourse processes* 25.2-3 (1998), pp. 259–284.

[5] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015).

[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[7] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (2017), pp. 354–359.