PREDICTING POPULARITY OF REDDIT POSTS

MIE1628

Abstract

The aim of this project is to predict the popularity of Reddit posts. To meet our goals, we structure the task into a multi-class classification problem and apply a series of feature engineering and machine learning methods to predict a post's popularity category. Our best model was a two-stage pipeline that yields a holdout set accuracy and F1 score of 71.9% and 67.5%, respectively. Feature engineering and data processing seem to be the limiting factors in improving performance of our model. To address these challenges, future work should aim to reduce the domain of posts analyzed, shrink the feature space through dimensionality reduction, and explore alternative regression-based and cascading classification pipelines.

Dylan Mendonca

Table of Contents

1 Introduction

Reddit is a popular social discussion platform where registered users submit posts that can be evaluated by other users. The platform is composed of themed sub-forums (or subreddits), which have their own rules that govern the content of posts and permitted topics of discussion. Users are given the opportunity to evaluate a post and express their opinion through votes (upvotes or downvotes) or comments.

The goal of this project was to apply feature engineering and machine learning approaches to predict the popularity of a Reddit post. Score (number of upvotes minus number of downvotes) is an important metric in determining a post's popularity and was the target variable in this project. The post's popularity is important to predict because it can influence a post's visibility on the website. For example, Reddit's home page often refers its users to trending or popular posts. A model that can predict popularity could be useful to users/institutions that want to create popular content, and to Reddit staff when deciding which posts to advertise on.

2 Dataset

Train and test datasets were provided by the course instructors for exploratory data analysis, feature engineering, and model development. A hold out dataset was provided one week prior to the presentation deadline to evaluate the model's ability to generalize to unseen data. All datasets had 57 unique columns (excluding the target variable score) detailing a post's content (title, thumbnails), attributes (creation timestamp, author), and location on the Reddit plaform (url, subreddit). The train, test and holdout sets contained information about 12525, 12556, and 14701 unique posts, respectively.

The target variable was score, which was a numerical, continuous integer value. The log-histogram of score (Figure 1) demonstrates that the target variable is heavily right skewed (and not lognormal), while the distribution plot of zero and non-zero scores (Figure 2) demonstrates that most scores (~60%) are zero. The statistical summary of scores (Table 1) further emphasizes the skew of the dataset toward low scores, given that the range of scores (583) is ~200 times larger than the IQR (3). The high imbalance in scores was an important consideration during model development as it would likely affect a model's ability to predict higher score posts.

We identified a strong correlation between the number of comments on a post and its score (Figure 3). This reason for this correlation was quite intuitive: posts with higher net upvotes are likely to have more comments because they can incite reactions from Reddit users. The number of comments played an important role in feature engineering.

The dataset appeared to be very non-linear, where most features did not have marked distinctions between high and low score posts. Figure 4 is a good example of this observation. While posts on restricted subreddits have lower scores, there are many posts on public and archived subreddits that also have low scores.

Table 1: Statistical summary of the score label

Mean	Std Dev	Min	25%	50%	75%	Max
7.5	29	0	0	0	3	583

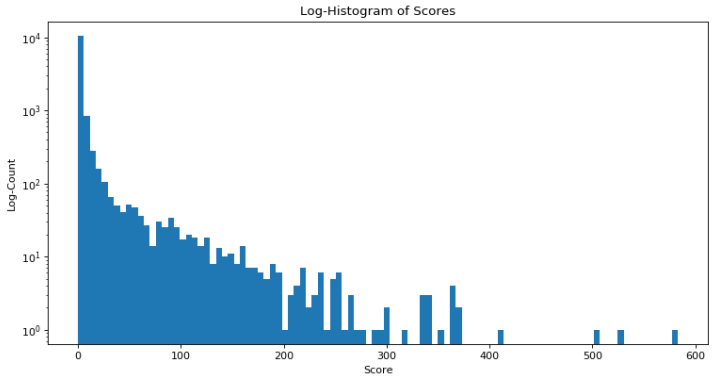

Figure 1: Log-Histogram distribution of the target variable, score

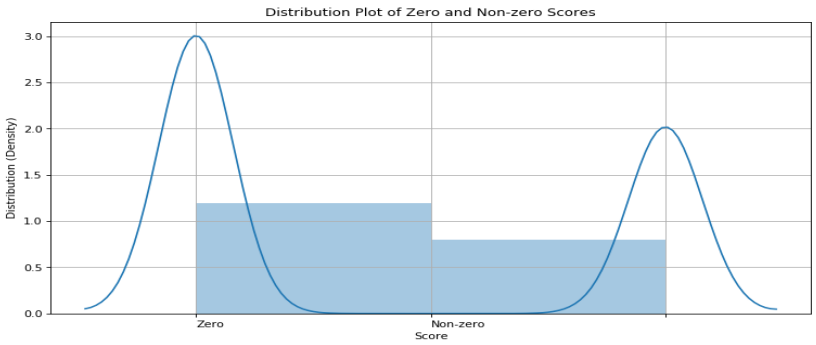

Figure 2: Distribution of zero and non-zero scores

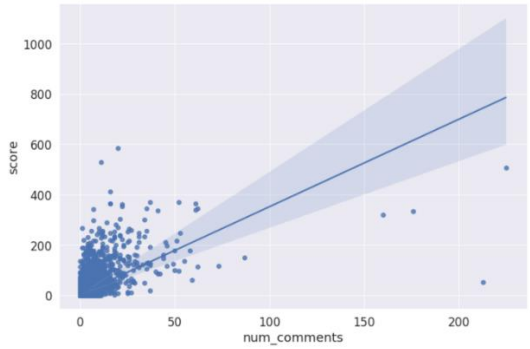

Figure 3: Scatter plot of score versus number of comments with line of best-fit overlaid

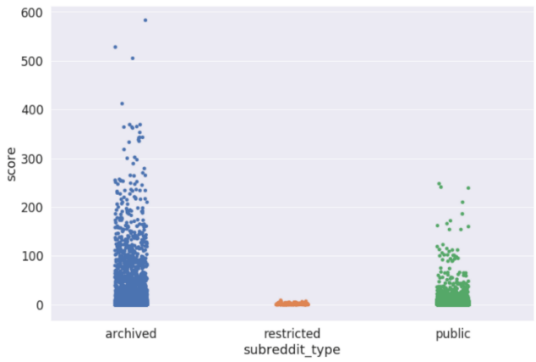

Figure 4: Relationship between score and subreddit_type

3 Approach

The subsequent sections contain the data preparation steps in the project. It should be noted that the holdout set was treated analogously to the test set mentioned in the subsequent sections. **The sections that I contributed to have been starred with an asterisk.** Apart from the sections discussed in the report, I also helped compile the final code into a notebook. Information about accessing the code is in Appendix A.

3.1 Outlier Detection*

We implemented an outlier detection system to remove some of the extremely high-score posts from our training set and narrow its range. To do so, we applied sklearn's EllipticEnvelope function [1]. A 2D Gaussian function was fit to a scatter plot of the number of comments versus score (Figure 3). An outlier fraction was specified and those points with the largest Mahalanobis distance from the mean (accounting for covariance) were assigned as outliers. The results of the model for 3 outlier fractions (1%, 5%, 10% going from left to right) have been included in Figure 5.

While this system was able to remove outliers from the training set, it was abandoned because the group was interested in converting the problem into a multi-class classification (as opposed to regression, which outlier filtering would have been useful for). Given that the dataset had a dearth of posts with high scores, we decided to retain all the data, bucket high score posts into a separate category, and predict their popularity in downstream machine learning pipelines.

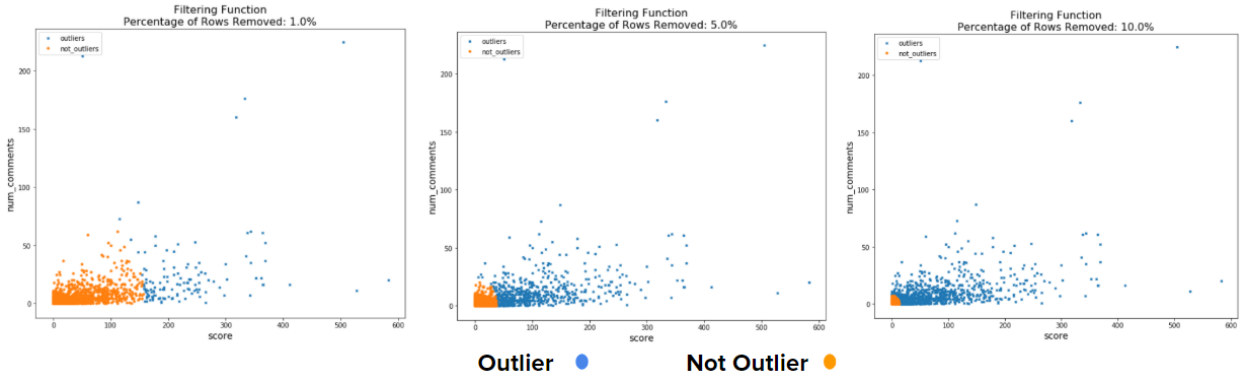

Figure 5: Applying the outlier function for different outlier fractions. From left to right: Outlier fraction set to 1%, 5%, 10%. The y-axis is number of comments and the x-axis is the score

3.2 Data Preparation & Feature Engineering*

The data pre-processing steps applied to the train and test sets have been outlined in Figure 6.

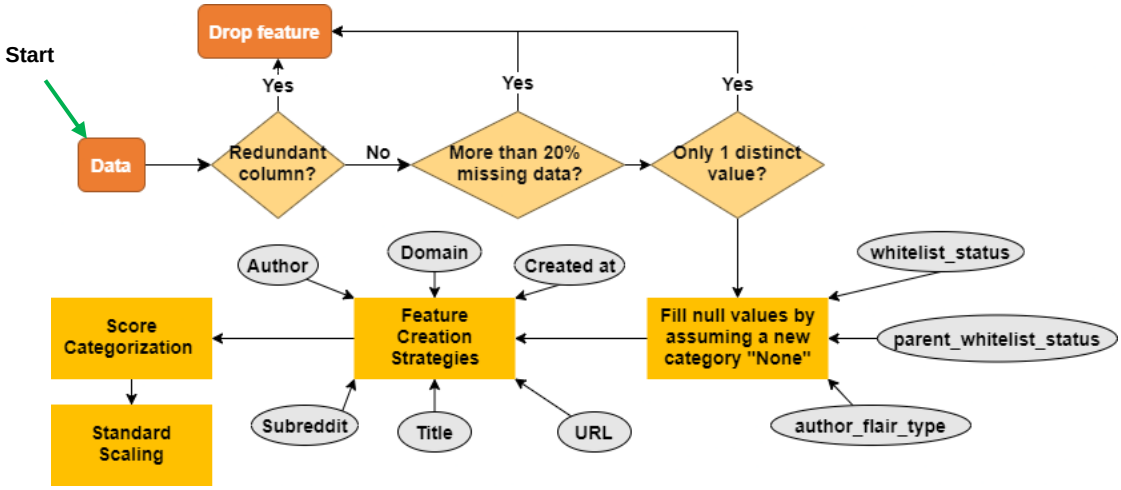

Figure 6: Data preparation and feature engineering flowchart

3.2.1 Data Preparation*

In the first step, 5 redundant columns were dropped. Columns that identified a post (id), had too many unique values (permalink), had very little information (thumbnail), or were already represented by other columns (subreddit_id and subreddit_name_prefixed were in the subreddit feature) were considered redundant.

We then dropped columns that had more than 20% missing data. The 20% threshold was chosen based on Figure 7, where a histogram of missing values in columns yielded two distinct groups – one group of columns with more than 88% of their data missing and another group of columns with less than 20% of their data missing. We dropped columns with a high percentage of missing values because there was not enough data to impute the missing rows. We selected a threshold of 20% but it should be noted that this threshold could be set to anything between 19.1% to 88.9%. We then dropped columns with 1 distinct value, because they would not be useful to downstream models.

Our datasets had 3 categorical columns with null values: whitelist_status, parent_whitelist_status, and author_flair_type. We created a new category for the missing values in these columns; instead of imputing them with the mode of the column. This was done because null (or missing) values in any of these columns meant that the post did not have that attribute present. For example, a null in author_flair_type meant that the author of the post did not have flair.

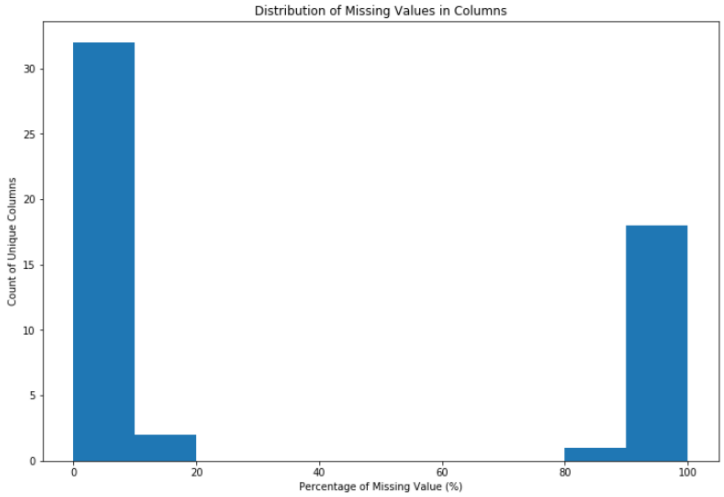

Figure 7: Count of columns by missing value percentage. Y-axis is the count of columns and the x-axis the percentage of missing values in the column

3.2.2 Feature Engineering Strategies*

The author, domain, created timestamp, subreddit, title, and URL features were converted into numerical and vector representations using a variety of techniques summarized in Figure 8.

3.2.2.1 URL*

The URL was cleaned and converted into a string of words separated by single spaces using regular expressions. This helped filter artifacts including, but not limited to, www., .com, .info, and ?param1=aparam2=b. A few examples of the URL cleaning have been included below:

1. http://www.generatorx.no/20060222/dreaming-in-images/ → generatorx dreaming in images
2. http://www.FaeLLe.com/2006/03/voodoopc-plans-8tb-media-pc.html → faelle voodoopc plans 8tb media pc

3.2.2.2 Title and Cleaned URL*

The title of the post and the cleaned URL (Section 3.2.2.1) were converted from strings into vectors using TF-IDF. The string features were tokenized by splitting words at spaces using PySpark's Tokenizer object. HashingTF was then used to create a 30-dimensional bag of words for the training set for each of the columns. We then applied PySpark's IDF object to weight different words based on their presence in the training set corpus. The trained TF-IDF model was used to transform the title and cleaned URL strings in the test set.

3.2.2.3 Created Timestamp*

The hour and day of creation were extracted from the created_at feature using PySpark's inbuilt timestamp extraction functions. We applied this technique because we believed that the time which a post is made makes a difference to its popularity because Reddit usage varies over the course of a day or week.

3.2.2.4 Subreddit, Author, Domain*

We postulated that popular subreddits, authors, and domains were more likely to have higher score posts. Based on this intuition, we decided to apply PySpark Window functions over the number of comments to compute

the average number of comments per author, subreddit, and domain. We chose number of comments because it was a strong indicator of popularity (Figure 1Figure 3). Window functions allowed us to convert these features into numerical variables that had good correlation with our target variable (Figure 9).

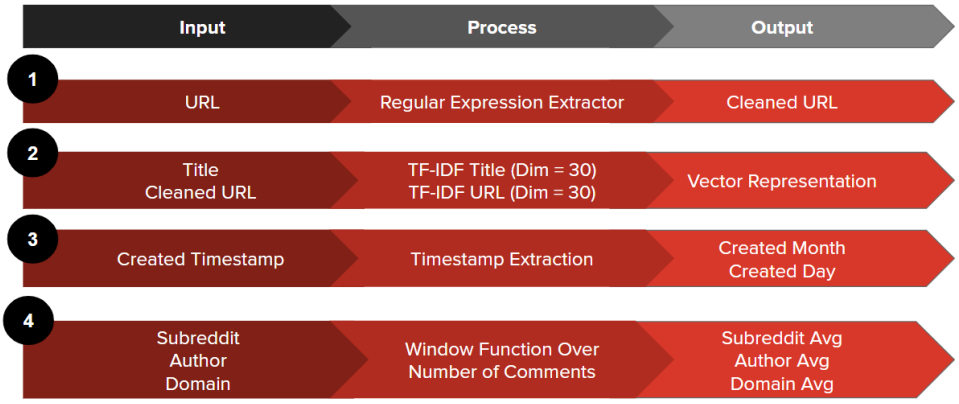

Figure 8: Feature engineering strategies on remaining columns

	score
subreddit_avg	0.096552
author_avg	0.392569
domain_avg	0.450686

Figure 9: Correlation between score and features engineered using Window functions

3.2.3 Encoding of Binary and Categorical Features*

One hot encoding was used to convert the remaining categorical features (eg: whitelist_status, parent_whitelist_status) into numerical variables using PySpark's StringIndexer and OneHotEncoder objects. One hot encoding was chosen because it was easy to implement and did not assign extra weights to labels based on their value. Binary features (eg: author_flair_type, over_18) were converted into 1s or 0s.

3.2.4 Standard Scaling*

All the feature columns were then scaled to have 0 mean and unit variance using PySpark's StandardScaler object. The object was fit to the training set and used to transform the test set. The scaling was done to normalize all features to the same range and help enable quicker and more accurate convergence of downstream linear models. The scaled features were not used in the downstream tree-based models.

3.2.5 Categorization of Score Feature

The task was converted into a multi-class classification problem by bucketing scores into categories. We decided to do this because of the heavy skew in score. Based on the skew, we inferred that the absolute difference in score becomes less meaningful as you move toward higher scores. For example, a post that has a score of 150 could be considered as popular as one that has a score of 100, because most posts have a score of 0. We also believed that categorization could offer other advantages, including: (1) better interpretability, (2) better model performance (because we generalize scores through categorization), (3) less effort normalizing scores and/or filtering outliers, and (4) ease of dealing with imbalanced data.

To categorize scores, a 10-component Gaussian Mixture Model (GMM) was fit to the training data and the intersection points between the components were selected as end points for categories. GMMs were chosen because they are robust and can approximate complex, non-linear distributions (as is the case with our skewed target variable) as a mixture of many Gaussians. The categorization and endpoints for each category

has been included in Table 2. Figure 10 shows that the GMM adequately bucketed scores because almost all categories (except for the first) are distributed relatively evenly.

Table 2: Buckets of scores and their ranges

Class	0	1	2	3	4	5	6	7	8	9
Score	0	1	2	3-4	5-6	7-9	10-16	17-37	38-133	>133

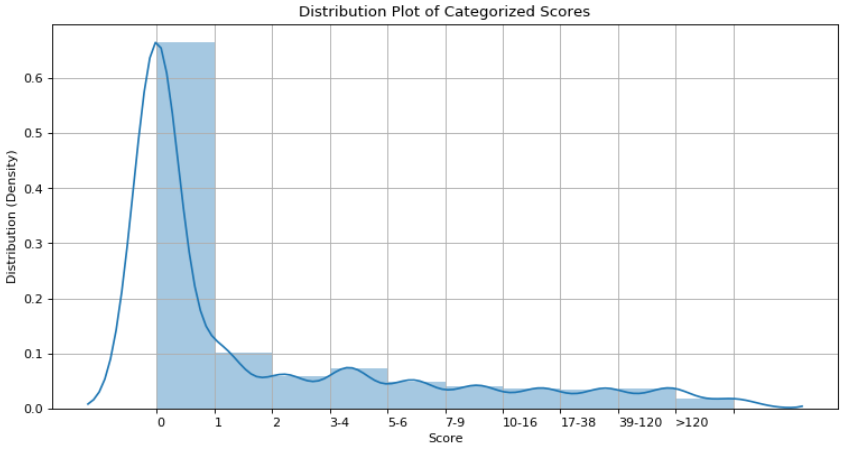

Figure 10: Distribution of bucketed score variable

3.2.6 Final Features

After the data preparation and feature engineering steps, we had 17 features which have been summarized in Table 7 (Appendix B). Accounting for the size of the vectors, each post had a 105-dimensional representation.

3.3 Model Development

3.3.1 Metrics Chosen

Macro-weighted F1 score (Equation 1) and accuracy (Equation 2) were chosen in the development, evaluation, and tuning of our models. F1 score was chosen because it accounts for both precision and recall in multi-class classification. The macro weighted F1 scores helped balance classes based on their prevalence in the dataset, allowing us to treat classes equally (useful because most of the dataset is 0). Accuracy was chosen because it gave an overall picture of the model's performance in predicting categories. In the model development stage, more precedence was given to the macro-weighted F1 score as it helped account for class imbalances better and incorporated more information (precision and recall) than did accuracy.

Equation 1: Macro-weighted F1 score

$$F_{wtd} = \sum_{j} F_j w_j$$

Where F_j is the F1 score for class j calculated using one-vs-all approach and w_j is the proportion of the j class

Equation 2: Accuracy score

$$Accuracy = \frac{\#\ of\ Correct}{Total\ \#\ of\ cases}$$

Area Under the ROC Curve (AUC) (Figure 11) was chosen to evaluate the performance of the stage 1 binary classifier (Section 3.3.3). AUC was used as the primary metric because it is scale and classification threshold invariant, and so provides an aggregate measure of the model's ability to distinguish between two groups.

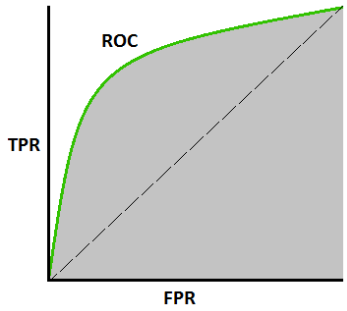

Figure 11: Calculation of AUC score. The y axis is the True Positive Rate and the x axis is the False Positive Rate. These values are calculated for many probability thresholds to obtain the Receiving Operating Characteristic (ROC) curve. The AUC is calculated as the shaded area [2]

3.3.2 Baseline Development

We tested linear and non-linear tree-based multi-class classification models for benchmarking purposes. In addition, the baselines allowed us to determine which model to implement in the two-stage pipeline discussed in Section 3.3.3. Logistic Regression (LR), Decision Tree (DT), and Random Forest (RF) models were trained and evaluated on the test data to predict the categorized score label variable. The hyperparameters for these baselines were set to the default (some of the hyperparameters are included in Appendix C8.3.

3.3.3 Two-Stage Pipeline

After baseline development, a two-stage pipeline was implemented to help deal with class imbalances and hopefully improve our prediction of the target variable. The first stage of the pipeline aimed to make a binary prediction on a post having a zero or non-zero score. To achieve this, we created a new column (score_g_zero) and assigned a 1 to posts with scores above 0 and a 0 to posts with scores equal to 0. The first stage was added in the hope that it would balance the score categories by filtering out posts with scores of 0.

The second stage of the pipeline took the posts that the first stage binary classifier predicted as 1 (i.e. scores above 0) and tried to do a multi-class classification of the final score category (which ranged from 0 to 9). We assumed that the more balanced dataset created in the first stage would help improve the predictive ability of the second-stage classifier. Predictions from both stages were combined to give a final set of results (we gave precedence to Stage 1's 0 prediction). Due to time constraints, we decided to use LR for both stages of the pipeline, because it performed the best on the test set during our baseline development.

3.3.4 Hyperparameter Tuning*

Hyperparameters were tuned at the end of the project for each stage of the pipeline using a combination of grid search and 3-fold Cross Validation (CV). 3-fold CV was used instead of 5-fold CV, because of computational limitations. The LR models were trained for 10 maximum iterations (as opposed to 30 or higher) to further address computational hindrances. The AUC and macro-weighted F1 score were used to tune hyperparameters for the first and second stages, respectively.

The hyperparameters tested have been included in Table 3. The Elastic Net Parameter governs the balance between absolute (L1) and squared (L2) regularization. A value of 0 is a purely L2 penalty and a value of 1 is a purely L1 penalty. The Regularization Parameter indicates the weight of the regularization term; the higher it is, the higher the regularization and chance to underfit. The fit intercept term indicates whether a bias is added. If False, the intercept of the linear decision boundary is forced through the origin and vice versa.

Table 3: Hyperparameters tested for stages 1 and 2 of the two-stage pipeline

Hyperparameter	Parameters tuned
Elastic Net Parameter	{0, 1}
Regularization Parameter	{0, 0.1}
Fit Intercept	{true, false}

4 Results & Discussion

The following sections present and discuss the few key results. All the results can be found in Appendix D.

4.1 Baseline Models

As stated in Section 3.3.2, the baseline models were used to guide the development of the two-stage classifier and to benchmark our model's predictive ability. Given that the 62.2% of the test set target variable was 0, the baseline results in Figure 12 were promising, as they performed better than a naïve classifier that predicts the majority class (this was also observed for train and holdout sets, which had 59.8% and 67.9% target zeros, respectively). LR outperformed DT and RF in both accuracy and F1 score on the test set. This trend was consistent for the train and holdout sets in all cases except two (DT and RF outperformed LR in accuracy on the holdout). Given that we placed more precedence on F1 score and that we did not have the holdout data available till later in the project, we selected LR as the model of choice for our two-stage pipeline.

It was surprising to see a linear model outperform the non-linear tree-based methods, given that our data was very non-linear (Figure 4). No experiments were done to investigate this phenomenon, but we postulate that the superior performance of the linear model arises from the engineering of non-linear features, which help "linearize" the problem, making it more amenable to the LR model.

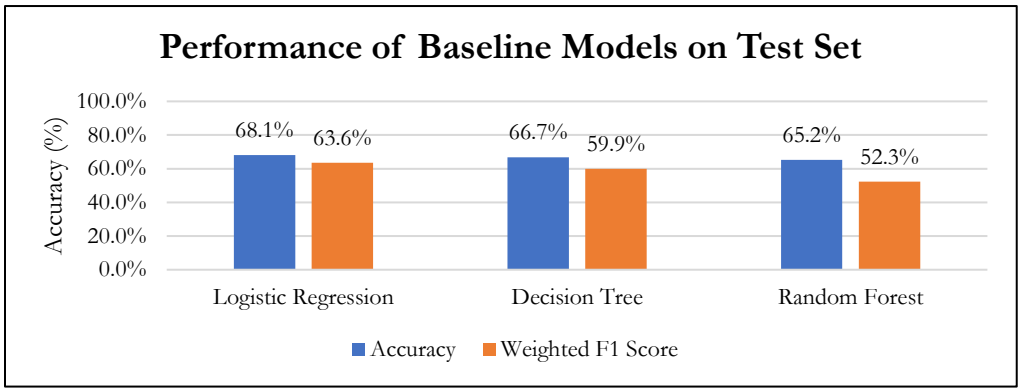

Figure 12: Performance of baseline models on the test set. The performance of the baselines on the test set was used to guide the development of the two-stage classifier

4.2 Two Stage Classifier

The two-stage classifier was built to help filter out the 0s and help create a balanced dataset for a downstream multi-class classifier. The binary classifier in the first stage performs very well, based on the high AUC and Precision (for 0 class) on the test and holdout sets (Figure 13). Figure 14 indicates that the Stage 1 model helps give a balanced distribution of score categories to Stage 2. Based on these results, we conclude that the binary classifier is an effective method of balancing the dataset.

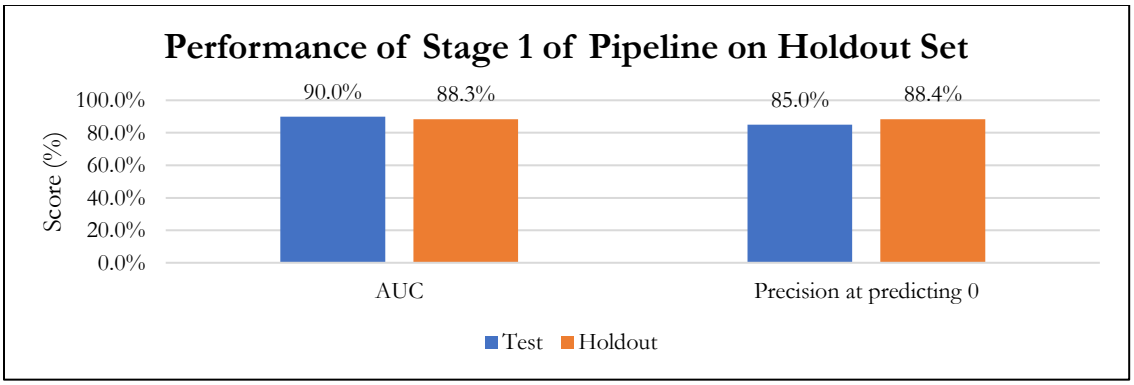

Figure 13: Performance of stage 1 of two-stage classifier on the holdout set

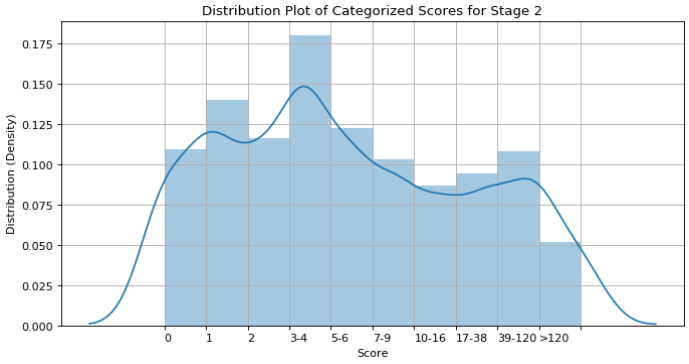

Figure 14: Distribution of score categories for data going into stage 2 for train set

From Figure 15, we can see that Stage 1 outperforms Stage 2 by large margins in both accuracy (difference of 29.6%) and F1 score (difference of 31%). The poor performance of the Stage 2 multi-class classifier compared to the Stage 1 binary classifier suggests that the model is having trouble generalizing to high score classes. This could be because of the sheer number of classes that need to be predicted (10 total classes), or the lack of data points within the original dataset with high scores. Furthermore, the poor performance of the second stage makes it apparent that a balanced dataset approach from the onset would not have been an optimal solution to the classification task.

Despite its low accuracy and F1 score, the Stage 2 classifier does help improve the predictions made in Stage 1, giving a boost of approximately 8.5% and 6% in accuracy and F1 score respectively for the combined predictions. Furthermore, the two-stage pipeline performs better than the LR baseline by 3.8% and 3.9% in accuracy and F1 score, respectively. These percentages, although small, are quite meaningful when scaled to even larger datasets. They also demonstrate that a cascading classification pipelines are a promising avenue for further research.

Based on the results from the train, test, and holdout data, it is quite unclear if we are over or under fitting. On one hand, the F1 scores in the tables in Appendix D suggest that we are not overfitting, because the train, test, and holdout performances are quite similar (further bolstered by the fact that the F1 score accounts for class proportions). However, Figure 16 suggests otherwise; because the accuracy difference between the two-stage model and the naïve classifier is much higher for the train set compared to the test or holdout sets.

It is quite apparent that we are underfitting to score categories that are higher than 0 and fitting well to score categories of 0, based on the poor performance of the Stage 2 classifier in comparison to the high AUC scores of the Stage 1 classifier. This could arise due to limitations in feature engineering, data processing, the number of categories, or the overall approach, because the second stage model could not generalize to a balanced dataset. More work needs to be done to validate the phenomena associated with over/underfitting.

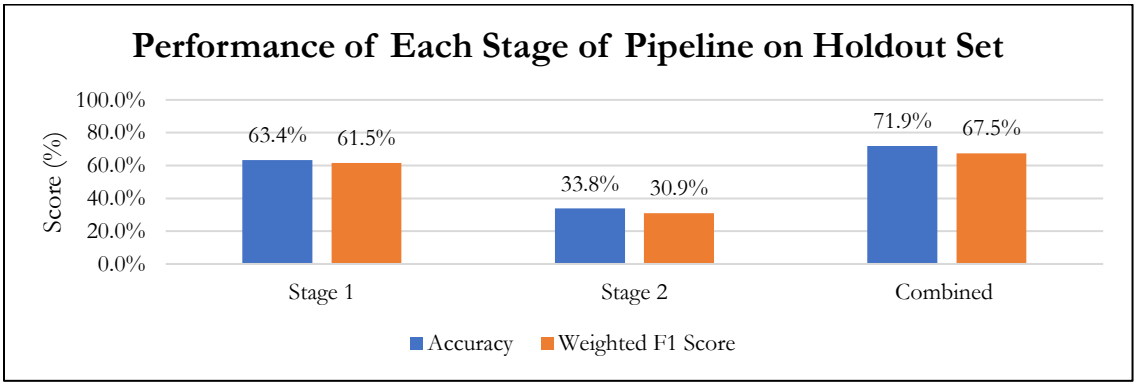

Figure 15: Performance of each stage of pipeline on holdout set

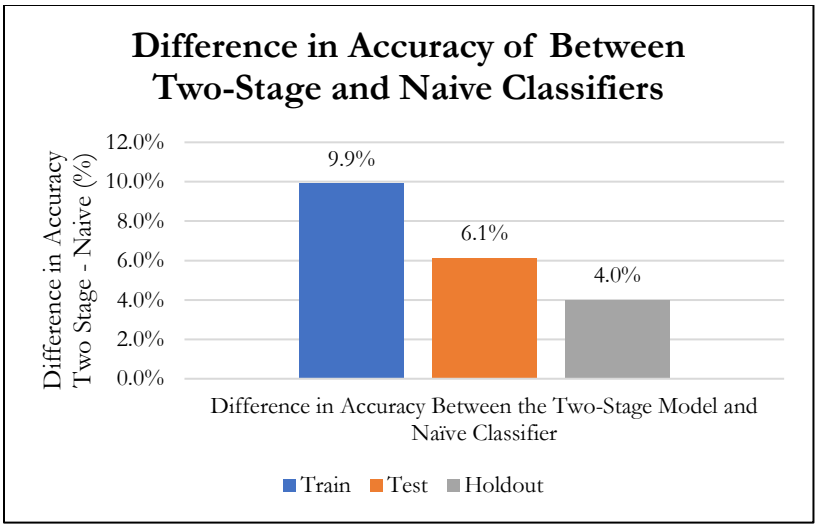

Figure 16: Difference in accuracy for the two-stage and naive classifiers for train, test and holdout sets. It is apparent that we may be overfitting because the improvement in accuracy of the train set is higher than the test or holdout sets

4.3 Hyperparameter Tuning

The hyperparameter tuning recommended that we use the default parameters for the Stage 1 classifier (Table 4), and that we only change the fit intercept to False (i.e. include a bias) for the Stage 2 classifier (Table 5). There was very little difference in performance between the tuned and untuned models (Table 6). The fact that tuning was not able to boost performance significantly further illustrates limitations in data processing/engineering and the overall model approach, as opposed to changing the parameters of the model. This experiment demonstrated that the models could not extract much more information from the dataset.

Table 4: Best hyperparameters for stage 1 based on 3-fold CV

Hyperparameter	Best hyperparameter
Elastic Net Parameter	0
Regularization Parameter	0
Fit Intercept	True

Table 5: Best hyperparameters for stage 2 based on 3-fold CV

Hyperparameter	Best hyperparameter
Elastic Net Parameter	0
Regularization Parameter	0
Fit Intercept	False

Table 6: Tuned Versus Untuned on Holdout Set

	Tuned	Not Tuned
Accuracy	71.4%	71.9%
Weighted F1-Score	67.0%	67.5%

5 Limitations & Future Work

Our work suggests that improving the ability to predict a post's popularity is very dependent on our features and data. I propose a few recommendations to tackle this issue. First, I recommend trying to narrow down the problem (and hence the dataset) to make it easier for models to generalize. This can be done in a variety of different ways. One example involves training models and making predictions on a dataset that has been restricted to a single domain or subreddit, instead of training and evaluating models on many different

subreddits (as we did in this project). This approach is promising especially if we consider the title column. Generating domain-level vector representations of the title would allow downstream models to generalize better, because the words that make a post popular in one subreddit might be completely different to the words that make a post popular in another subreddit.

Another data-related approach that could simplify the problem would be to lower the number of score categories. Our GMM assumed 10 components; no work was done to show why 10 components was the best choice. The superior performance of the Stage 1 binary classifier compared to the Stage 2 multi-class classifier demonstrated that there may be a correlation between a reduction in number of categories and model performance. While the same results might not hold when the number of categories increases above 2, this option still seems like a promising avenue for further investigation.

One approach that should be considered is the dimensionality reduction of the feature set. Each post was represented by a 105-dimensional vector, which was very burdensome computationally and might have resulted in a lot of data sparsity. More work should be dedicated to reducing the number of features, through dimensionality reduction techniques (PCA, TSNE), or by selecting only the most important features (feature importance from LR, correlations). This might help alleviate the computational burden and battle data sparsity.

Aside from the data and features, there are a few recommendations I can make from a model perspective. One promising concept involves extending the binary Stage 1 classifier to a series of cascading binary classifiers, where each subsequent classifier is doing a binary prediction of the next score category (i.e. first classifier does a 0/0+ prediction, the second classifier does a 1/1+ prediction, and so on) [3]. In addition, future researchers might consider other embedding models like Word2Vec, BERT, or GLoVe, which can help capture contextual dependencies between words in a corpus far better than a count-based system like TF-IDF [4]. This approach would be especially useful when converting titles to vectors. Lastly, a fundamental assumption of this work was structuring the problem into a classification task. Hence, future work should attempt a regression-based approach and assess whether this assumption is valid.

There were many computational and time constraints for the project. This led to the use of 3-fold CV (instead of 5-fold CV) with a lower number of maximum iterations (10 iterations) than might have been necessary for the model to converge. Furthermore, it meant that we could not further investigate other models or hyperparameters (eg: the threshold for the binary classification in Stage 1 of the pipeline, the embedding size of our TF-IDF vectors). To address these challenges, I recommend subsampling the dataset and moving the work to a larger computational cluster, where more experiments can be conducted.

6 Conclusion

This report aimed to predict the popularity of Reddit posts. To do so, we structured the problem into a multi-class classification, and applied feature engineering and machine learning techniques to classify a post into a popularity category. We developed linear and non-linear baseline classifiers, of which Logistic Regression performed the best, with a holdout set accuracy and F1 score of 70.1% and 67.2%, respectively. We then applied a two-stage classification pipeline, where Stage 1 does a binary classification that filters 0 scores, followed by a Stage 2 multi-class classification that predicts the range of score categories. This approach performed better than the baselines, yielding a holdout set accuracy and F1 score of 71.9% and 67.5%, respectively. The results suggested that feature engineering and data processing is the limiting factor in improving performance on this task. Hence, I recommend that future work should be dedicated to reducing the domain of posts analyzed, shrinking the feature space through dimensionality reduction, reducing the number of score categories, and experimenting with alternative regression-based and/or cascading classification pipelines.

7 References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[2] S. Narkhede, "Understanding AUC-ROC Curve," Medium, 26 6 2018. [Online]. Available: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. [Accessed 26 8 2020].

[3] J. Gama and P. Brazdil, "Cascade Generalization," *Machine Learnin,* vol. 31, no. 3, pp. 315-343, 2000.

[4] V. Gangadharan, D. Gupta, T. Athira and T. A. Athira, "Paraphrase Detection Using Deep Neural Network Based Word Embedding Techniques," in *2020 4th International Conference on Trends in Electronics and Informatics*, Tirunelveli, 2020.

8 Appendices

8.1 Appendix A

The code has been published on DataBricks under the following URL: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/8668619575683126/1420393744458190/5219188826199367/latest.html

8.2 Appendix B

Table 7: Final feature set

No	Feature Name	Description	Type
1	brand_safe	Whether the subreddit is safe for advertisers	Binary
2	can_gild	Whether the post can is gilded	Binary
3	is_crosspostable	Whether the post is crosspostable	Binary
4	no_follow	Whether the post has follow enabled	Binary
5	over_18	Whether the post is restricted to over 18 audiences	Binary
6	author_flair_type	The flair type of the author	Binary
7	parent_whitelist_status	The whitelist status of the parent post	Encoded Category
8	whitelist_status	The whitelist status of the post	Encoded Category
9	subreddit_type	Type of subreddit	Encoded Category
10	created_hour	Hour of creation of post	Encoded Category
11	created_day	Day of creation of post	Encoded Category
12	num_comments	Number of comments on post	Numerical
13	subreddit_avg	Average number of comments for the subreddit	Numerical
14	domain_avg	Average number of comments for the domain	Numerical
15	author_avg	Average number of comments for the author	Numerical
16	title	Title extracted as 30-dim vector using TF-IDF	Vectorized string
17	cleaned_url	Clean URL extracted as 30-dim vector using TF-IDF	Vectorized string

8.3 Appendix C

Table 8: Hyperparameters for Logistic Regression baseline

Maximum Iterations	30
Elastic Net Parameter	0
Regularization Parameter	0
Fit Intercept	True

Table 9: Hyperparameters for Decision Tree baseline

Maximum Depth	5
Maximum Bins	32
Criteria	gini

Table 10: Hyperparameters for Random Forest baseline

Maximum Depth of Tree	5
Maximum Bins	32
Impurity	gini
Number of Trees	20

8.4 Appendix D

Table 11: Baseline Accuracy Results (best model highlighted in green)

	Train	Test	OOT
Logistic Regression	69.3%	68.1%	70.1%
Decision Tree	68.4%	66.7%	71.7%
Random Forest	62.4%	65.2%	70.5%

Table 12: Baseline F1 Results (best model highlighted in green)

	Train	Test	OOT
Logistic Regression	64.5%	63.6%	67.2%
Decision Tree	61.5%	59.9%	65.5%
Random Forest	48.7%	52.3%	59.2%

Table 13: Performance of Stage 1 Classifier

	Train	Test	OOT
Accuracy	60.7%	60.9%	63.4%
Weighted F1-Score	55.1%	56.5%	61.5%
AUC	90.6%	90.0%	88.3%

Table 14: Performance of Stage 2 Classifier

	Train	Test	OOT
Accuracy	40.0%	31.8%	33.8%
Weighted F1-Score	37.3%	28.4%	30.9%

Table 15: Combined performance of two-stage classifier

	Train	Test	OOT
Accuracy	69.7%	68.3%	71.9%
Weighted F1-Score	65.5%	62.9%	67.5%