DS-1001 Final Report

Real-Time Predictive Analytics for High Scoring Reddit Posts

Ross Bernstein, Brian Kelly, Jatin Khilnani and Xiaoyi Zhang

December 2018

Abstract: This project focuses on analyzing real-time submission data scraped from Reddit, in order to predict viral posts. The problem was framed as a binary classification problem in which a Reddit post was considered to be viral if it had a score equal to or greater than a certain threshold 24-hours after the post was created. In order to predict which posts will become viral, we employed techniques such as model stacking, and various sampling methods along with logistic regression, random forest models, countvectorizer, TF-IDF, and Bernoulli Naive Bayes. Reddit submission data was scraped continuously for several days. In addition to hyperparameter tuning and feature engineering, character-embedding of the titles was stacked onto the classification models to further improve model performance. The models provide reliable predictions of potentially viral posts, laying the foundation for building a robust and computationally-efficient pipeline in a real business setting.

Contents

1 Business Understanding

Websites that post viral content, also referred to as “content-aggregators”, generate most of their revenue from advertising related to internet traffic.[1] These aggregators post content that was taken from somewhere else, often other content aggregators. If a certain news story, funny video, or cat-picture has proven to generate views on another site, then it is likely to succeed again. This is a very competitive game, as these businesses compete to re-post viral content first. The importance of posting such content as early as possible is based on the nature of what it means to be “viral”. Definitions of virality vary, but all emphasize that it is not just a high view-count that matters, but rather interaction—specifically, how many times something is shared and reposted. As a given link spreads through content aggregators and social sharing, the growth in views become exponential and explodes. Much like a pyramid scheme, those who shared the link earliest get the most traffic and ad-revenue. It is therefore very valuable to be able to predict the virality of content before other sites do. This project aims to design a classification model to detect viral content on Reddit by five minutes after the post is made.

Reddit is one of the most successful content aggregators, and their users also post a lot of original content. This makes them one of the most common starting-points for viral content that is often found on other aggregator websites, and an ideal source of data for our model.

Reddit, a website that is divided into a series of topic-specific groups called subreddits, allows registered users to post content, comment on posted content, and like or dislike posted content and comments. Posts that have a high score (number of upvotes minus downvotes) appear towards the top of a given subreddit’s front page.

Reddit itself could benefit from a model that predicts the popularity of their posts, by slightly adjusting the way the price ads. They currently use a CPI scheme, or ”cost-per-impression”, which means that advertisers simply pay for the number of times their ad is seen. Utilizing our classification model, they could consider charging for ads to just appear on specific posts, with pricing that correlates to how likely that post is to go viral. Some advertisers may be willing to pay a premium for this service.

Lastly, the ability to predict virality would also benefit companies that maintain an active online presence and directly interact with the Reddit community through comments. As shown in Figure 1, Max Woolf shows that early comments on a viral post have a higher probability of becoming a top comment on a given post.[2] Having the top comment on popular posts means greater exposure and this also allows a company’s ”message” to be heard by more people.

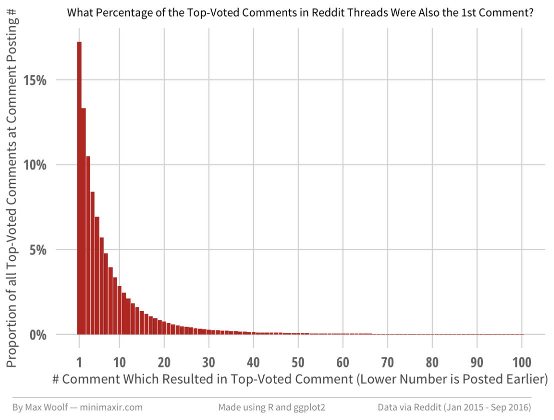


Figure 1: Woolf shows that the earliest posts often become the top comments on Reddit posts.[2]

2 Data Understanding

2.1 Data Collection & Extracted Features

We obtained Reddit submission data in two ways. Historical data was obtained from the Pushshift database, which was used for exploratory analysis. Real-time submission data was scraped using PRAW (Python Reddit API Wrapper) .[3][4] The script utilizes the API's SubredditStream class, which yields a continuous stream of new submissions for a given subreddit. For each new submission, all the relevant features are extracted and written to a csv file where each row contains a unique submission (indexed by a unique id). At the same time, the script also continuously looks up these already-retrieved submissions at regular intervals, and requests updated information for the fields that vary over time (such the score, and the number of comments made, among others). These updated fields are then stored in a separate csv file, with each row representing a submission at a given point in time. The following fields and subreddits were scraped:

1. Fields at time of posting:
   - `id, author, title, subreddit, time created, url, over_18, time_created`
   - `comment_karma, link_karma` (author's past comment and post performance)
   - `is_self` (if post contains body-text or is just a link)
2. Fields updated over time:
   - `score`
   - `num_comments` (number of comments on post)
   - `num_crossposts` (number of times the same submission was made to other subreddits)
   - `stickied` (post was pinned to top of page)
   - `gilded` (someone has paid money to give the post a visible gold star)
3. Subreddits:
   - `AskReddit, Pics, Gifs, Videos, WorldNews, Funny, Aww, Gaming`

We wanted to track the progression of each submission's fields, starting from the time of posting, until 24-hours later. Since the early performance for each submission is most important for our model, and the overall progression was for exploratory analyses only, we pulled submissions at smaller intervals early on and wider intervals later. The specific intervals we used were (in minutes): 5, 10, 15, 30, 60, followed by every hour thereafter until 24 hours. Since every post is created at a different time, we needed to update the fields of each post relative to the time the post was made. In order to do this, we used a PriorityQueue() that ensured we were checking each submission at the correct point in time.

The speed of operation, and the rate at which we could pull information from Reddit, was a primary concern for the success of this script, since anything that interfered with pulling the correct fields at the correct time would invalidate our data. Reddit imposes a rate-limit on API requests of about 1 request/second. Initially, this was a big problem until we found a way of requesting submissions in batches of up to 100 at a time. With regards to the processing speed of the script on our end, we tried several techniques to ensure smooth operation. Since the script is performing two functions at once (getting new submissions and updating the fields for old ones), we first used threads to carry out each function. Ultimately, we switched to multi-processing because we were warned that the API isn't thread-safe, and multi-processing performed better anyway.

We ran the scraper continuously for about a week, although at one point it lost connection, and caused us to lose data for about 2 days. Ultimately, we were still able to track about 200,000 unique submissions.

2.2 Exploratory Analyses

The decision of which subreddits to scrape was made by making a number of considerations. First, we ran a script using the Pushshift data that extracted every front page submission for 100 days straight. This allowed us to determine which subreddits appeared on the front page the most (See Figure 2). We also considered the rankings of subreddits by subscriber-count. Finally, we took the subreddits appearing in both of these lists and plotted the number of posts scoring above 100 that appeared on each of their front pages to gauge popularity. All three plots helped us to make an educated guess.

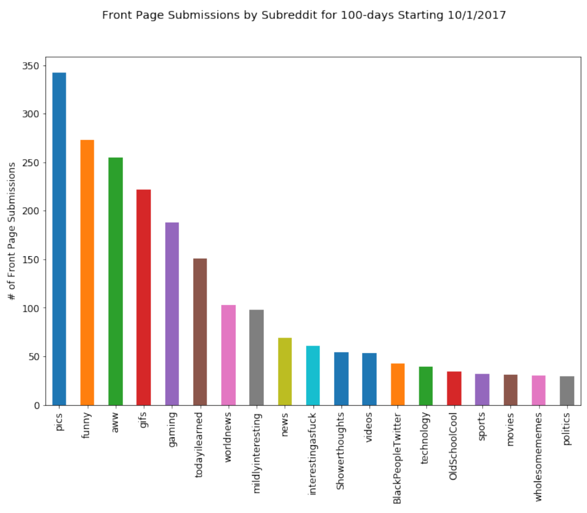


Figure 2: Front page submissions by subreddit for 100 days

Figure 3 shows the AUC with and without our time-dependent features such as score, and num_comments. This plot shows that while the score at 5 minutes is our strongest predictor, the model is not entirely dependent upon it, as one might expect.

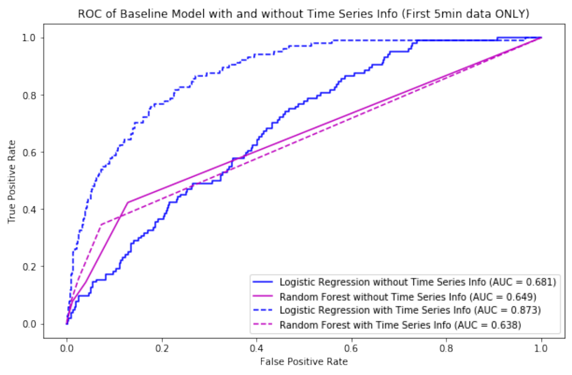


Figure 3: ROC for logistic regression and random forest with and without time-dependent features, such as score and num_comments

Figures 4 and 5 show different visualizations of submission-score progressions over time. These plots give us some important insights. As one would expect, the viral post-scores grow exponentially over time. The scores start out with gradual growth very early, increase very rapidly, and then eventually flatten out. What is interesting is that the curves are very smooth. The highest ranking submissions each day are seen in the section of Reddit called 'top'. Reddit also contains 3 sections known as 'new', 'rising', and 'hot' that allow users 3 different ways of filtering submissions. When submissions are first made, they start out in 'new'. The amount of votes a score receives here is crucial to it's success. If the submissions doesn't score high enough in this early stage, it will be buried by newer posts. However, if it scores high enough during this early stage, it will show up in 'rising'. The 'hot' section utilizes an algorithm for deciding what submissions are shown based on a combination of score and the age of the post. Submissions on 'hot' have scored high recently, meaning there is currently a lot of activity, or "buzz", surrounding these posts. The exact mechanism by which this algorithm works is hidden to the public. The 'hot' section serves to bring a lot of quality submissions into the spotlight that might have otherwise gone unseen if users only ever saw the top scoring posts of the day. It is also Reddit's way of "evening the odds". Based on our domain knowledge of Reddit, we believe that this has the effect of both smoothing the curves and causing them to converge. Otherwise, it is likely that we would see even less viral submissions, with even higher scores.

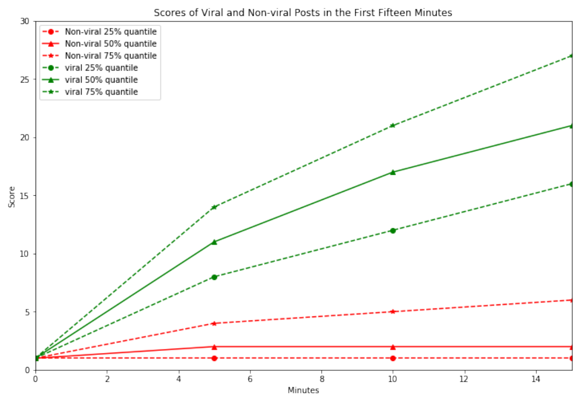


Figure 4: The post scores for the first, second and third quartiles of viral and non-viral posts in the first fifteen minutes.

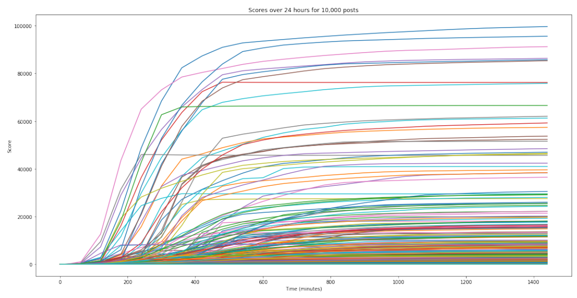


Figure 5: Plot of the scores for 10,000 posts over a 24 hour period

Figure 6 contains a heatmap we generated which shows, for each hour and day, the number of viral posts created during that time. This gives an idea of the best times to post on Reddit in order to maximize the chances of going viral. Note that our heatmap is missing a few days because of our scraper losing connection to Reddit once during the week. We did not want to include partial data in this plot, so we left those days out. However, based on our prior research, we expect that the visible trend seen on the days we do have, would extend to the missing days. The heatmap shows that most viral posts tend to be created in the morning, a moderate amount are created throughout the day, and the least are created late at night. It is possible that people tend to browse Reddit most during their morning commute. Posting earlier in the day also generally allows more time for a post to gain upvotes. This led us to add 'day' and 'hour' as features in our model.

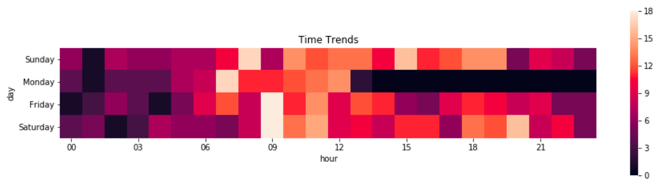


Figure 6: Viral Posts by Day and Hour Posted

2.3 Selection Bias

All post data with NaN values, which only occurred in the post scores, were dropped. In this fashion, models only used data from posts monitored continuously for 24 hours, ensuring that there are no partial observations that could lead to bias. It was initially tempting to include posts for which we did not have complete data, but which exceeded our viral-threshold before 24 hours. This would have introduced bias because we would be rejecting other posts that didn't reach the threshold before 24 hours. We don't really know what would have happened to the scores of those posts if given the full 24 hours, so classifying posts in this fashion would have introduced a selection bias towards posts that achieve virality early.

Despite our efforts to minimize selection bias, the choice of several subreddits over all posts due to computational constraints introduces bias. Moreover, for business reasons, the subreddits chosen are among those subreddits that have the highest rate of new posts, which ignores the plethora of subreddits that have much more limited activity. The chosen subreddits may have content-specific trends that do not generalize to the overall population. For instance, viral posts on the subreddit "AskReddit", a collection of questions answered by the Reddit community, will have very different content than those on the subreddit "aww", a set of adorable pictures and videos that tends to focus on animals.

Just as the choice of subreddit creates bias, the selection of a limited time period also presents potential selection bias. For computational reasons, the analyzed post data was restricted to several days of posts, which means that the collected data may not represent the population of posts over longer time periods. The limited time period causes models to learn seasonality trends as properties of the general population rather than periodic variations.

3 Data Preparation

3.1 Target Variable & Feature Engineering

The first step in preparing our data set was to join the 2 files that our scraper produced, creating additional columns to store the time-series values for each time interval (i.e. the score_5, score_10, score_15). Next, we set our target variable. Figure 7 shows a plot of the 95th through 99th percentile of submission scores over the course of one month (October 2018). There is a sharp increase at the 99th percentile, which corresponds to a score of 3656. We chose to base our target variable on this value—if the score of a post surpassed this value at 24 hours, then we considered it to be a viral post. We then set our 'predict-by' time, which is the time interval by which we want to predict virality. Setting this value equivalently meant that we dropped every time-dependent column after this time. In our case, every column that was scraped after 5 minutes was dropped.

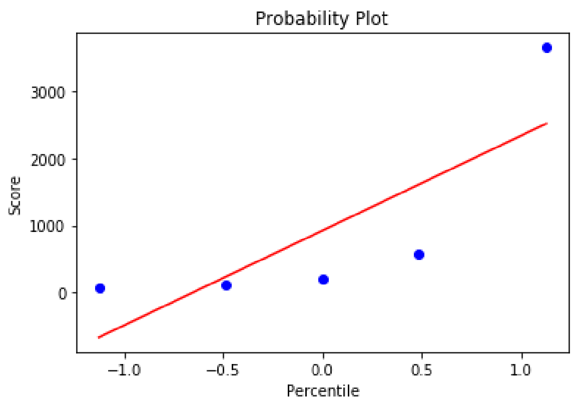


Figure 7: A plot of the 95th through 99th percentiles of submission scores.

Based on our existing fields we extracted:

- The hour of day and day of week each post was made
- The length of the post title
- The domain of the link given in the post (reddit if not an external link)

For our 'title' column, we used text embedding and trained a separate model on just the titles. We tried both CountVectorizer and TfidfVectorizer for text embedding, along with Bernoulli Naïve Bayes for this model, with TfidfVectorizer performing best. The predictions from this model were added as a feature in our original model. Thus we employed model stacking by combining this into our original model.

3.2 Sampling & Scaling

Because of our heavily skewed data, several sampling and scaling techniques were compared evaluated on our models. We compared StandardScaler(), MinMaxScaler(), class weighting, and downsampling our majority class on both of our baseline models.

4 Modeling and Evaluation

Given the threshold for virality discussed in the previous section, the analysis of Reddit submissions became a binary classification problem based on numeric features. Logistic regression and random forest, straightforward and computationally inexpensive methods, were the main algorithms used in modeling. Since the title of the posts are very short, character-based embedding was stacked with the two methods mentioned above to improve model performance. The natural language processing model captured predictive sub-word level information in the Reddit post titles.

4.1 Baseline Models

The default sampling and parameter settings in the Python SkiKit-Learn package for logistic regression and random forest were a baseline for choosing optimal sampling methods and hyper-parameter tuning. The comparisons were performed using ROC-AUC metrics. The performance of baseline models are shown below in Figure 8.

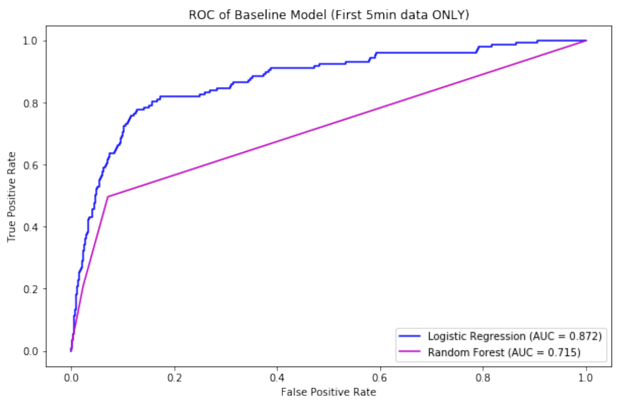


Figure 8: ROC of Logistic Regression and Random Forest with default settings. Training data is sampled randomly without balancing between target variables.

The baseline models give decent AUCs, which implies that both algorithms are decent choices and exploration of more complex was unwarranted. To further improve the models, the following factors were considered: 1) balanced sampling of the training set; 2) hyper-parameter tuning and bias-variance trade-off; and 3) feature selection based on importance. Moreover, the models were evaluated to determine whether overfitting occurred, and feature-correlation was examined to ensure that the model results were reproducible.

4.1.1 Metrics Selection

In this task we use AUC as a metric to evaluate our models, since it is a comprehensive measurement over false positive/negative predictions, revealing more information than using accuracy as a metric.

4.1.2 Sampling Techniques

Since the distribution of the target variable is highly skewed such that only 1% of the data are positive instances, we hypothesized that balancing the positive and negative target variables would yield a better result. We examined several different methods for correcting the data skewness. The training set was downsampled such that the ratio of positive to negative instances was 1 : 1. The 'class-weight-balanced', a built-in parameter in sklearn to automatically apply weighting to the classes that are inversely proportional to class frequency. The 'stratify' parameter was set to true, which ensured that the testing set maintains the actual ratio of target variables.

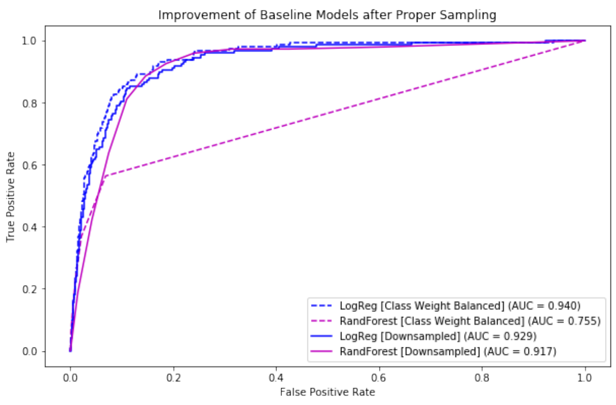


Figure 9: ROC of logistic regression and random forest after downsampling and class weight balanced

As shown in Figure 9, the downsampled training set gives a more consistent improvement with both algorithms than that of the 'class-weight-balanced' parameter. We also noticed that the feature importances made much more sense with downsampling. Downsampling was used in all future hyper-parameter tunings.

4.1.3 Evaluate Overfitting

Though the models are not complicated, overfitting remains a risk due to small sample size after downsampling. Ten-fold cross-validation on the baseline models was performed and the results of four randomly chosen folds are shown in Figure 10.

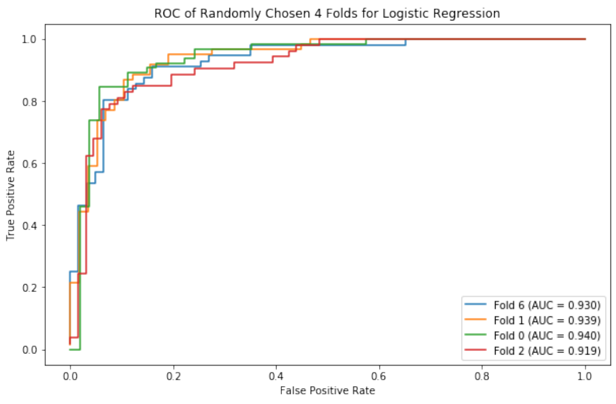


Figure 10: ROC of randomly-chosen four folds among 10-fold cross-validations on baseline logistic regression.

Since the ROCs of the four folds have significant overlap, the sample size should not cause the models to overfit.

4.2 Logistic Regression: Hyper-Parameter Tuning

The logistic regression hyper-parameters C, the inverse of regularization strength, and p, the norm of regularization term, were tested over a broad range of values for C. The results of the grid search for both $L1$ and $L2$ penalties are shown in Figure 11.

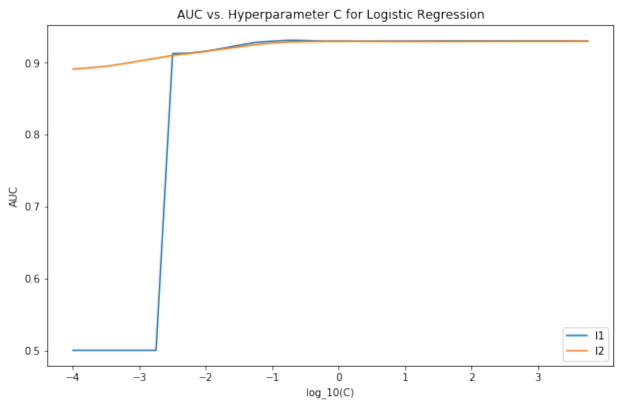


Figure 11: Performance of logistic regression on 10-fold validation set with $C \in \{10^{\frac{i}{4}} : i \in [-16, 16]\}$, both $L1$ and $L2$ penalty.

Figure 11 shows that when $C >= 0.01$, the performance of both $L1$ and $L2$ regularization converge at a high AUC where a maximum is obtained at $C = 0.1$. The performance on the testing set for $C = 0.1$ is shown in Figure 12.

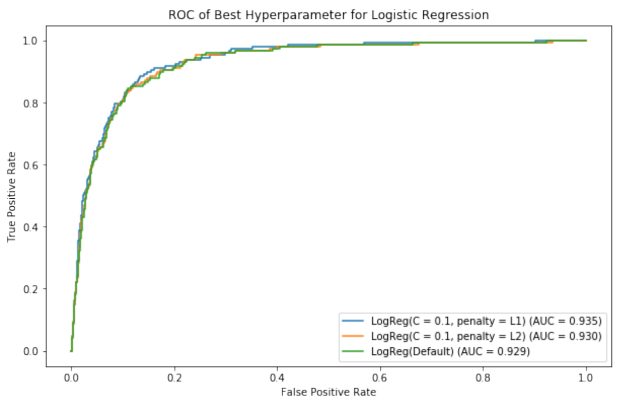


Figure 12: ROC of logistic regression for $C = 0.1$ except for the curve labeled with 'default'

Figure 12 shows a slight improvement of the model after parameter tuning. The $L1$ penalty outperforms $L2$ on the testing set. The $L1$ penalty shrinks unimportant features to zero, which may explain the better performance given the large number of features.

4.3 Random Forest Classifier

Random forest can be viewed as a combination of multiple decision trees. In particular, it trains on different parts of the same training set, with the goal of reducing the variance and preventing overfitting, which is one of the major drawbacks of decision trees[5]. Moreover, while decision trees use Hunt's algorithm, a greedy model that optimizes the decision at each node but not necessarily reaches the global optimum, random forests' random subset of features limit the error due to bias and variance[6].

4.3.1 Bias-Variance Trade-off

Similar to logistic regression, we test on a hyper-parameter grid consisting of a wide range of 'min-samples-split' and 'min-samples-leaf', which are the minimum number of samples to split a node and the minimum number of samples to be at a leaf node, respectively. A more complex model has low values of both hyper-parameters. The results of the hyper-parameter grid search with 10-fold validation is shown in Figure 13.

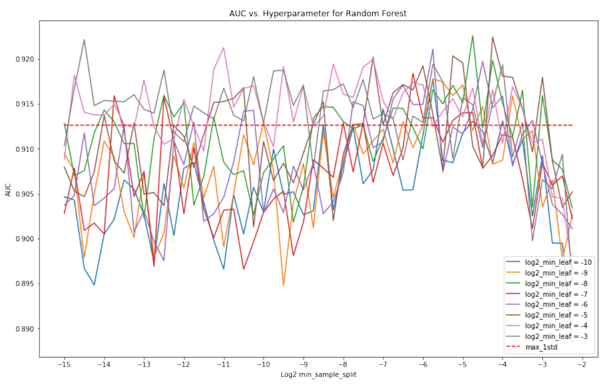


Figure 13: AUC of Random Forest given 'min-samples-split' $\in \{2^{\frac{i}{4}}, i \in [-60,-8]\}$, 'min-samples-leaf' $\in \{2^i, i \in [-15,-2]\}$, 10-fold cross-validation. The red dashed line marks mean - 1st standard deviation of the highest AUC in the grid.

The bias-variance trade-off was considered in choosing the optimal configuration of 'min-samples-split' and 'min-samples-leaf' using the one-standard-error rule. In this case, the simplest model whose AUC surpassed the red dashed line in Figure 13. Specifically, the MSE of k-th fold cross validation is given by

$$MSE_k(\Theta) = \frac{1}{n_k} \sum_{i \in F_k} (y_i - \hat{f}_{\Theta}^{-k}(x_i))^2$$

where Θ is the hyperparameter set, n_k is the number of samples in the k-th fold. The equations for the sample standard deviation and standard error are shown below.

$$SD(\Theta) = \sqrt{Var(\{CV_j(\Theta)\}_{j=1}^{k})}$$

$$SE(\Theta) = \frac{SD(\Theta)}{\sqrt{K}}$$

Initially $\hat{\Theta} = \arg\min_{\Theta} CV(\Theta)$ and the value of Θ moves in the direction of increasing penalty until the ineuality $CV(\Theta) \leq CV(\hat{\Theta}) + SE(\hat{\Theta})$ is violated. In this way, we create an optimal trade-off between minimizing the variance without substantially increasing the bias. Based on the grid search results, 'min-samples-split' and 'min-samples-leaf' was set to 2^{-5} and 2^{-3}, respectively.

The performance of random forest also depends on the number of estimators (i.e. number of trees). As shown in Figure 14, the performance of the model increases drastically with the number of estimators at the beginning. After 100 estimators (the default value), the AUC slowly diminishes and converges to a slightly lower value.

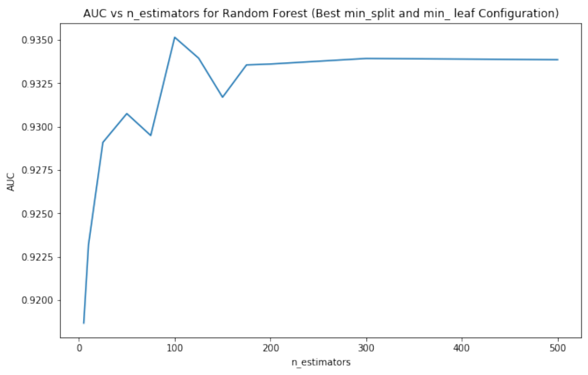


Figure 14: The estimator number in a random forest model was varied where 'min-samples-split' $= 2^{-5}$ and 'min-samples-leaf' $= 2^{-3}$. For each estimator number, 100 epochs are run to find the average.

4.3.2 Feature Selection

The results of calculating the feature importance for the model parameters in a 10-fold cross-validation are shown in Figure 15. The feature importance in each fold varies, which suggests that training and validation data should be split prior to feature importance evaluation. Moreover, the observed feature importance values align with the real world meaning for the parameters. Besides time series features, inherent properties of a post, including the subreddit category, subscriber number, and whether the post contains text, already gives significant information about whether a post will become viral.

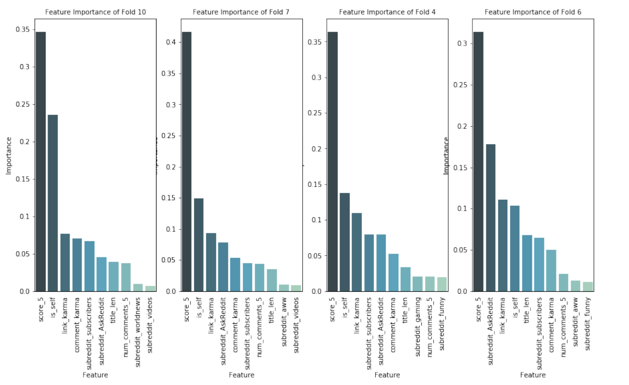


Figure 15: The top 10 features for randomly-chosen four folds among the 10-fold cross validation. Gini impurity is used for defining the feature importance.

In order to determine the model performance of among different combinations of parameters, 100 epochs were run for every feature number in the candidate pool, using the best hyper-parameter configuration chosen in the previous section.

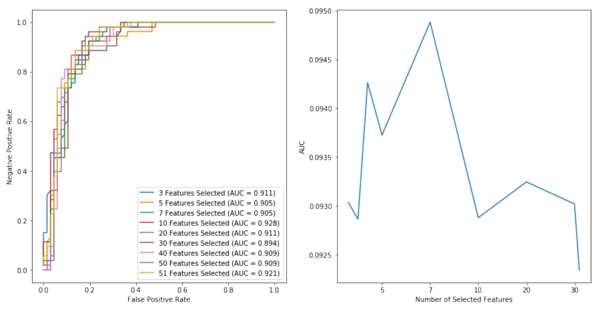


Figure 16: Left: ROC of Random Forest using different feature numbers in a randomly chosen epoch. **Right:** average of AUC given by varying feature numbers over 100 epochs.

Figure 16 shows that the model performance drops as feature number goes beyond seven, reaching the lowest AUC when in a model that uses all features. Since feature importance is dependent on the sample from the data set, the number of model parameters was chosen instead to specify which features to use. The performance of the random forest classifier on the testing set is shown in Figure 17.

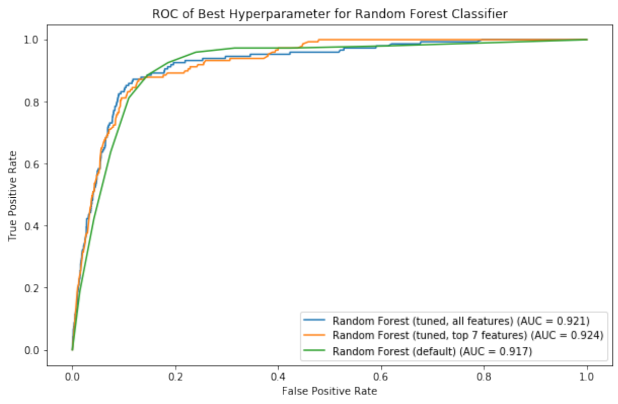


Figure 17: ROC of the random forest model where 'n-estimators' = 100, 'min-samples-split' = 2^{-5}, and 'min-samples-leaf' = 2^{-3}.

The AUC of the three curves in Figure 17 shows that the hyper-parameter tuning is effective, and that a model with only the top seven features can further improve the model performance as tested on validation set.

The results of the model selection and testing are summarized in Table 1.

Table 1: The computed AUC of several instances of logistic regression and random forest models

Model	**AUC**
Logistic Regression (baseline)	0.872
Logistic Regression (downsampled, default hyper-parameters)	0.929
Logistic Regression (downsampled, C = **0.1**, penalty = **L1**)	0.935
Logistic Regression (downsampled, C = **0.1**, penalty = **L2**)	0.930
Random Forest (baseline)	0.715
Random Forest (downsampled, default hyperparameters)	0.917
Random Forest (downsampled, 'min-samples-split' = 2^{-5}, 'min-samples-leaf' = 2^{-3}, **ALL features**)	0.921
Random Forest (downsampled, 'min-samples-split' = 2^{-5}, 'min-samples-leaf' = 2^{-3}, top **7 features**)	0.924

4.4 Model Explanation

For deeper insight into the models, one may want to know we look into how the features play in our models' final prediction. Here we analyze the feature correlation and explanation (by LIME package).

4.4.1 Feature Correlation

Correlated features not only have no positive effect on our model, but also lead to risk in multicollinearity, which causes numeric unstability especially in Logistic Regression[7]. Here we plot the feature co-variance matrix to check the correlation between important features selected by Random Forest.

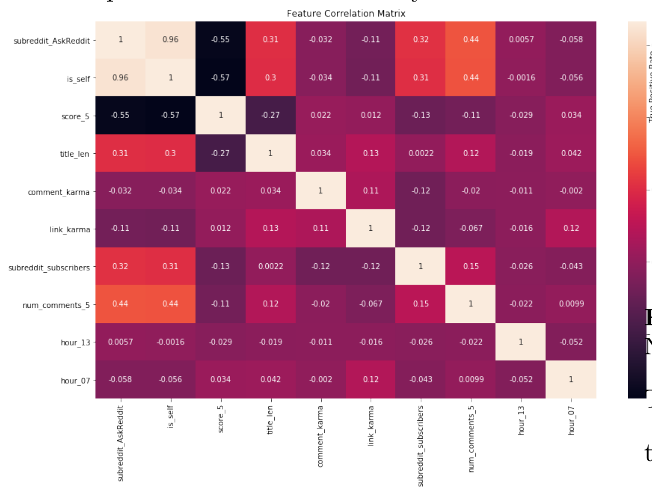


Figure 18: The co-variance matrix of top 10 features selected by Random Forest model.

Most correlations amongst the important features are close to zero, which shows that feature engineering is successful in preventing redundant information. Moreover, we obtain business insights from the co-variance matrix. For example, feature 'subreddict-Askreddit' is positively correlated to 'is-self' with a coefficient of 0.96, which implies that most posts under Askreddit are original. Also, the score at 5-th minute after posting is negatively correlated with 'is-self'with a coefficient of -0.57, which tells that posts able to gain popularity early are likely to contain contents shared from other sources.

4.5 Extension: NLP and Model Stacking

As the raw data set contains the title of each post, we want to test whether viral posts have distinguishing textual features, under the intuition that some key words may occur very often in the title of popular posts. We tested count vectorization and Tf-Idf on Logistic Regression and Bernoulli Naive Bayes and the result is given by following:

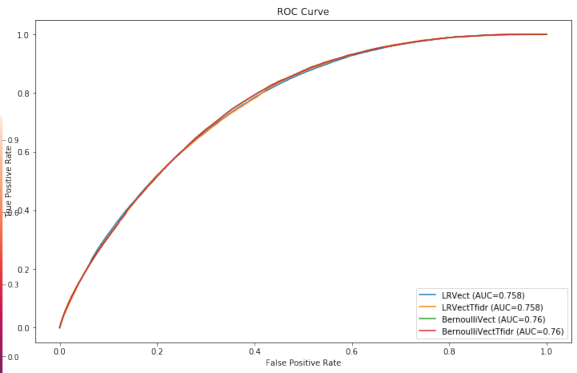


Figure 19: ROC of Logistic Regression and Bernoulli Naive Bayes on post titles.

The AUC scores show that models using only textual features do not beat our baseline. Options to improve the performance includes applying more advanced classifications algorithms (such as RNN, CNN, LSTM, etc.). Since Logistic Regression and Random Forest already give AUC higher than 0.9, we consider it does not worth using neural networks or deep learning in general, which have limited room for improvement in this case but can be significantly more time consuming. However, inspired by neural networks, which are essentially compositions of optimization functions, we propose a novel model that stacks the NLP task and classic classification algorithms. Specifically, for each post x_i, we first use Tf-Idf vectorization to extract textual information from its title, then apply Bernoulli Naive Bayes to predict $\mathbb{P}(y_i = 1|\ BernoulliNB(title\ of\ x_i))$. This probability is then used as a feature for Logistic Regression and Random Forest models as selected in the previous section. Moreover, in this case we care more about sub-word level information and hence using character-embedding instead of word-embedding, because user-generated contents contain many short-hands and emoticons. Our results are plotted as following.

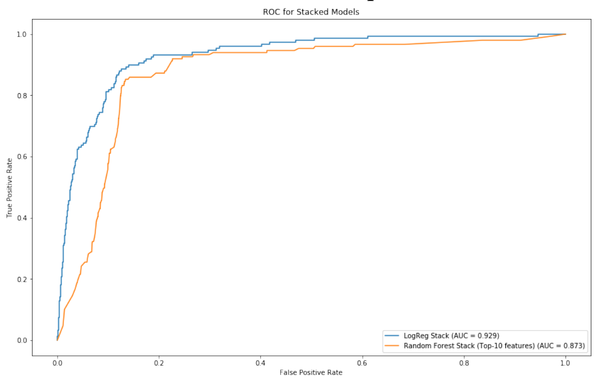


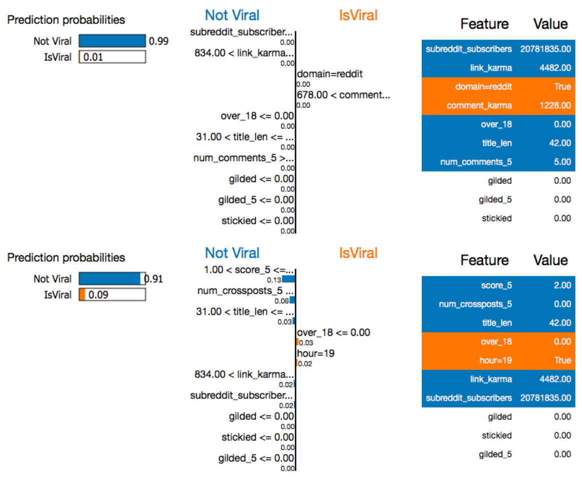
Figure 20: ROC curves of stacking tri-gram character-embedding onto Logistic Regression and Random Forest, whose hyperparameters are determined in the previous section.

Unfortunately model stacking has not significantly improved our models. For future study, we plan to try more advanced embedding methods that can take context information into account (such as Word2Vec).

4.5.1 Deeper Look into the Features

In addition to the analysis of feature importances and their correlation with each other and to the classification problem, we have also tried to make the model transparent through implementation of LIME (Local Interpretable Model-Agnostic Explanations)[8] package to explain the results as suggested. The foundation behind the explanation this package provides, is that it is a local linear approximation of the model's behaviour. While the model may be very complex globally, it is easier to approximate it around the vicinity of a particular instance. While treating the model as a black box, LIME perturbs the instance to be explained and learns a sparse linear model around it, as an explanation.

During the process of model evaluation, the package helped in identifying model performance and tune it accordingly, as per the figure below.

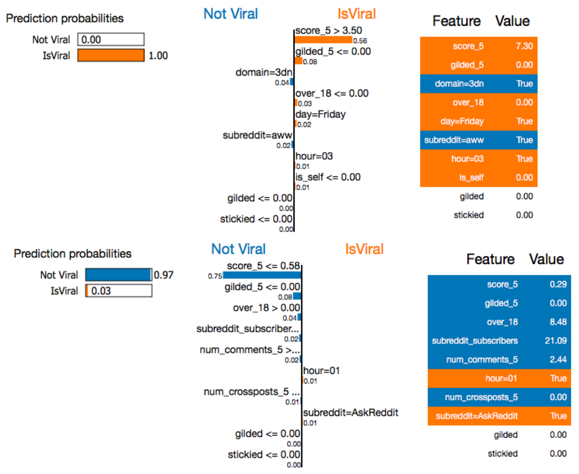


Feature	Value
subreddit_subscribers	20781835.00
link_karma	4482.00
domain=reddit	True
comment_karma	1228.00
over_18	0.00
title_len	42.00
num_comments_5	5.00
gilded	0.00
gilded_5	0.00
stickied	0.00

Feature	Value
score_5	2.00
num_crossposts_5	0.00
title_len	42.00
over_18	0.00
hour=19	True
link_karma	4482.00
subreddit_subscribers	20781835.00
gilded	0.00
stickied	0.00
gilded_5	0.00

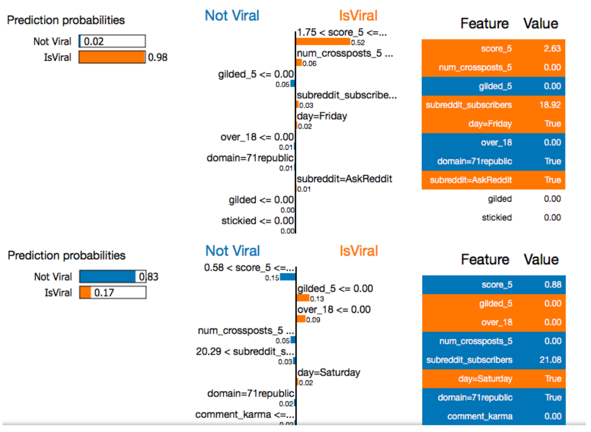
Figure 21: Identify Model Differences. Top: Naive Logistic Regression. Bottom: Baseline Version.

For the purpose of explaining our best-fit predictive classifier Logistic Regression, we have taken a pair of positive and negative cases (both true & false) to be illustrated by the package and provide transparency to its functioning.



Feature	Value
score_5	7.30
gilded_5	0.00
domain=3dn	True
over_18	0.00
day=Friday	True
subreddit=aww	True
hour=03	True
is_self	0.00
gilded	0.00
stickied	0.00

Feature	Value
score_5	0.29
gilded_5	0.00
over_18	8.48
subreddit_subscribers	21.09
num_comments_5	2.44
hour=01	True
num_crossposts_5	0.00
subreddit=AskReddit	True
gilded	0.00
stickied	0.00



Feature	Value
score_5	2.63
num_crossposts_5	0.00
gilded_5	0.00
subreddit_subscribers	18.92
day=Friday	True
over_18	0.00
domain=71republic	True
subreddit=AskReddit	True
gilded	0.00
stickied	0.00

Feature	Value
score_5	0.88
gilded_5	0.00
over_18	0.00
num_crossposts_5	0.00
subreddit_subscribers	21.08
day=Saturday	True
domain=71republic	True
comment_karma	0.00

Figure 22: Results Explanation. A: True Positive. B: True Negative. C: False Positive. D: False Negative.

These findings are in-line with the data understanding gathered through exploratory analysis, e.g. timing of post is an important factor in it going viral. Thus, indicating the reliability of the model and its results. Continued monitoring in production would enable proactive model enhancement.

5 Deployment

A company could theoretically deploy our model in its current form, since both our scraper and model have high performance and accuracy. However, depending on the company's needs, some modifications to both the model and the script might be required. For one, whereas we stored the submission data in csv files and read them manually into a jupyter notebook for processing and model-creation, the deployed version would feed the submissions directly into the model for classification or probability estimates. Code could be added that would alert the user when the model predicts a viral post so they can evaluate it for reposting, or it could be set up to automatically submit the content of the post to a website, requiring no human interaction at all. It would be beneficial to also store all of the submission data, so the model can continually improve itself by periodically training on the updated data set.

5.1 Technical & Process Considerations

The solution in its current form follows the guidelines and best principles from modeling and coding perspective, and hence attains good quality and results, both in performance and accuracy measures. As illustrated in prior sections, the data mining solution comprises of three parts.

1) Data Extraction: A data scraper (Python script) that captures real-time updates to posts on regular intervals 2) Data Processing and Model implementation: Executed through a set of modular Python methods in Jupyter notebook 3) Result visualization and explanation.

In order for the solution to be work on industrial level, following steps have to be taken:

1) Drill down on the target business/client use case for which any tweaking to the model may be required, e.g.

- Focus on viral prediction versus quality post identification (at time zero), both or something else;
- Metric verification - Precision vs Recall - its implications to the target stakeholders.

2) Evaluate and communicate the required production ecosystem capability (e.g. computational resource) that supports models to perform in maximal efficiency.

3) Perform necessary modifications to the model and code to make it compatible to various tasks:

- Tune hyper-parameters based on specific business

problem;
- Automate codebase, create pipelines for the scripts to run continuously with minimal manual intervention;
- Setup framework for audit, load balance, control, error handling and notifications to check for things like missing data that may lead to bias, data corruption, script failure/blockage due to upstream/downstream/environment issues, continuous enhancements to the model;
- Develop user-friendly interface/communication mechanism for transferring the results to the stakeholders for early action.

5.2 Ethical Considerations

The ability for aggregator websites to predict and link to popular posts shares many ethical concerns with other online social media platforms. Like individuals, these companies choose what content to post in the hopes of attracting internet traffic. The ability to predict which posts become viral and blindly posting the content without regards to the topic has the potential to allow hateful messages and other harmful content, such as spurious news and information, to be reposted. These aggregator websites have a responsibility to curb content that may be harmful to others and instead favor neutral to positive content.
Aggregators have also historically been known to steal content from sites without proper accreditation. Plagiarism is both illegal and unethical. However, the immense scale of aggregator content makes laws rather difficult to enforce. This places the burden on aggregator websites to enforce responsible reposting.

6 Appendix

6.1 Code

Github repository: https://github.com/xiaoyizy/RADS
Python packages:
- lime[9]
- tldextract[10]

6.2 Team Work

Ross Bernstein: Reddit scraper, exploratory data analysis, data preparation, feature engineering, sampling techniques.
Brian Kelly: Formatted Pushshift data, exploratory data analysis, feature engineering, model selection, LaTex formatting.
Jatin Khilnani: Data preparation (validation & visualization), model evaluation (accuracy & performance), code modularization, LIME implementation.
Xiaoyi Zhang: Model selection, hyper-parameter tuning, feature importance, NLP and model-stacking.

References

[1] Max Woolf. Predicting the Success of a Reddit Submission with Deep Learning and Keras, June 2017.

[2] Max Woolf. What Percent of the Top-Voted Comments in Reddit Threads Were Also 1st Comment?, November 2016.

[3] Bryce Boe. PRAW: The Python Reddit API Wrapper, 2017.

[4] Jason Baumgartner. PushShift IO, November 2018.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, second edition edition, 2008.

[6] Thomas Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40:139–157–, August 2000.

[7] David A. Belsley. *Conditioning Diagnostics: Collinearity and Weak Data in Regression*. Number Wiley Series in Probability and Statistics in Volume 262. Wiley, 1991.

[8] Marco Tulio Ribeiro. Local interpretable model-agnostic explanations (lime), 2016.

[9] Marco Tulio Ribeiro. API reference for lime, 2016.

[10] John.Kurkowski. tldextract 2.2.0, 2018.