Predicting Popularity of News Articles Using Machine Learning

Huseyin Altinisik and **Chuxuan Yang**
{hualtinisk,soyang}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

This project focused on using machine learning to predict the popularity of a given piece of news article using existing datasets. In this project, we tested three regression models: `Linear Support Vector Regression`, `Random Forest Regressor` (RFR), and `Lasso Regression`. After conducting a series of experiments using these models, we ranked their performances based on the results of news article popularity prediction. We found that `LinearSVR` performed the best, with an r2 score of over 0.8. This was true for both articles from "Moreover" and "Opoint", our two major data source. The best model used a combination of grid search with cross-validation. We think this project can be used to support the media and entertainment industry's efforts in producing contents that generate high volumes of readers, and can help many businesses better understand readers' interest.

1 Introduction

About fifty years ago, reading in paper copies through subscriptions was the only choice for many families. However, as technology advances, digital reading has gained tremendous popularity among the younger generations. According to the Fortune Magazine, today 85% adults in the US read news on their mobile devices. Therefore, it is in news medias' commercial interests to know which of their articles generate the most attention from the readers.

For this project, we set out to use machine learning techniques to predict popularity of a news article based on its text content. Our dataset contains over 12000 articles from different websites, covering a wide range of topics. Some articles have a "momentum" score, which signals its popularity among readers. The momentum scores range from 0 to 6448.06; for this project, we assumed that the higher the momentum score, the more popular the article. To turn our source data into usable feature data for this project, we performed text data pre-processing using `pandas`, as well as feature extraction using `TfidfVectorizer` from the `scikit-learn` library(Pedregosa et al. 2011). We then trained our models using `scikit-learn`'s built-in `LinearSVR`, `RandomForestRegressor`, and `Lasso Regression` models on articles that carry momentum scores. We will explain in greater details for each step in later sections, including our results in model performances.

In the next section, we will talk about preparing our data prior to training them on any models.

2 Background

Data Pre-Processing & Feature Extraction

The raw data for this project are stored in two large parent gzip files, "Moreover" and "Opoint". The Opoint parent gzip file originally contained 2679 child gzip files, whereas the Moreover parent file contained 12420 child gzip files. Each child gzip file stores an ".xml" file. Inside each ".xml" file we found a different number of articles, including meta data such as its unique article ID, the author, the source of the article, as well as the article content, which we considered to be our feature for this project. We also had a csv file called "Momentum" that stores the article IDs as well as its matching momentum scores for some of the articles from our file source.

Our first task was to iterate over each article inside the folders and the ".xml" files ,and consolidate all articles with a momentum score into a new CSV file for model training purposes. To do so, we iterated over all articles inside the Moreover files and extracted the contents for articles with an "id_article" that can be found in "Momentum.csv". For the Opoint files, we iterated over the files and saved contents for articles with a matching "feed_article_id" that exists in the Momentum csv file. We realized after finishing this step that our output csv file had a few empty or unusually structured rows; and since our data size is large enough, we decided to discard those rows.

Because text data cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length, we decided to perform feature extraction using the `TfidfVectorizer`. The `TfidfVectorizer` takes in individual articles in string formats, tokenizes the article strings into words and gives an integer id for each possible token or word by using white-spaces and punctuation as token separators. The vectorizer then counts the occurrences of tokens in each article and essentially normalizes the dataset.

Through this process, each individual token occurrence fre-

quency (normalized or not) is treated as a feature, and the vector of all the token frequencies for a given document is considered a multivariate sample.

Feature Modification

We realized at the early stages of our project that for large article texts, some words will be very present (e.g. the, it, a in English) hence carrying very little meaningful information about the actual contents of the article. If we were to feed the data on frequency directly to a model, those very frequent terms would potentially shadow the frequencies of rarer yet more interesting terms. Hence, we defined a list of stop words prior to running any models. Our list of words looks like this: ['the', 'a', 'is','are','for','that','as','it','to','be']. Later in our experiments, it occurred to us that our CSV files had many duplicated articles with the exact same contents. For example, our sample size before we deleted any duplicates for articles from Moreover was 2354, and after we dropped all samples that have the same titles, we had only 1164 samples left. We will discuss our results later in this paper.

3 Models & Experiments

For this project, we decided to use the following three models:

Linear SVR

Short for Linear Support Vector Regression, this model is similar to the Support Vector Regression model, with the parameter kernel = 'linear'. According to `scikit-learn`, `LinearSVR` is implemented in terms of liblinear rather than libsvm, hence it has more flexibility for the choice of penalties and loss functions, and can scale better to large numbers of samples.

- Hyperparameters
  1. C parameter (default = 1.0)
     For this model, we first ran 10-folds cross validation on the C parameter, which stands for penalty for the error term. We implemented the model with L2 penalty. Since the bigger the C value, the less regularization is used, we expected to see an improvement in model performance as C increased.
  2. loss (default = L1)
     The default setting for the loss function is L1, the "epsilon-insensitive" loss. After getting poor cross validation performance from this setting, we changed the loss setting to be L2, the "squared epsilon-insensitive". Following this tuning, we immediately saw a significant increase in our model results.
- Result Metric
  For LinearSVR, we looked at the r2 score as an indication of model performance. The r2 score is a statistic measure that tells the fitness of a model. A score of 1 indicates that the regression line perfectly fits the data.

Random Forest Regressor

This model creates a diverse set of classifiers by introducing randomness in the tree construction process. Because of such randomness, we expected the bias and variance of this model to be slightly higher. The model fits a number of decision trees on the sub-samples of the data set, and uses averaging to improve accuracy scores and to prevent over-fitting. Therefore, we expected it to yield an overall well-performed model.

- Hyperparameters
  1. n_estimators (default = 10)
     This represents the number of trees in a forest. We ran CV on this parameter first with values $[5, 10, 15, 20, 25]$, then with $[50, 100, 150, 200, 250]$, on top of the 10-folds cross validation we also performed on the model.
  2. max_features (default = n_features)
     This indicates the number of features to consider when looking for the best split. For our project, we first set max_feature to be the number of features we have for each article: the number of columns in our data frame. We also tried setting it to be 'sqrt' and 'log2', in which case max_features=sqrt(n_features) and max_features=log2(n_features).
  3. max_depth (default = None)
     This parameter represents the maximum depth of the tree. We left this setting as it was, because we wanted for all the nodes to expand till all leaves are pure.
  4. min_samples_split (default = 2)
     This stands for the minimum number of samples required to split an internal node. We also used the default setting of 2.
- Result Metric
  The performance of the Random Forest Regression was also measured using r2 scores.

Lasso Regression

Lasso is a regularization technique that works by penalizing the magnitude of feature coefficients to minimize the amount of error between the predictions and actual observations. Lasso performs L1 regularization, which is the sum of the weights in the loss function. Lasso shrinks the less important features coefficient to zero, which works well for feature selection when we have a huge number of features as in our case.

- Hyper-parameters
  1. alpha
     The alpha parameter is a constant value that is multiplied by the L1 term. The higher the alpha value, the stronger the penalization. If the alpha value is 0, the Lasso regression will result in the same coefficients as a linear regression (Albon 2017). The alpha values we tuned for this projects were $[0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5]$.

- Result Metric
  The Lasso Regression performance was calculated using the r2 scores.

In the next section, we will discuss our results for the models.

4 Results

Our results are displayed in Figure 1. For `LinearSVR`, the

Model Settings	Moreover	OPoint
LinearSVR, L1	−0.059	−0.034
Linear SVR, L2	0.826	0.860
RFR, max_feature = 'None'	0.125	0.373
RFR, max_feature = 'sqrt'	0.263	0.350
RFR, max_feature = 'log2'	0.096	0.337
Lasso regression	0.554	−0.435

Figure 1: Results under 10-folds Cross Validation

r2 score improved substantially after we switched from L1 to L2 penalty. We think this is because the L2 penalty led to lower regularization strength. For the different C values we tested, 1.5 turned out to be the one that gave us the best result among all.

For `RandomForestRegressor`, we started out using a smaller number of trees, ranging from 5 to 25 trees. However, our initial r2 scores from `RFR` yielded extremely high variance, ranging from 0.249 to −4.88. We realized that this was correlated to the smaller number of trees we had, so we increased our tree size substantially. Our results can be seen from the figure above, and a number of 150 and 200 trees seemed to consistently generated the best results.

Overall, our models still had relatively high variance in results. Under the 10-folds cross validation we performed for `RFR` and `LinearSVR` for example, we saw that the r2 scores could still range from as low as −0.257 to as high as 0.989. Figure 2 below visualizes one example of such variance. We thought this was because we had way too many features to deal with in this project, and this substantially increased the randomness of the results. For both `RandomForestRegressor` and `LinearSVR`, the results are also low-biased. We assumed the high variance, low bias combination happened because there were too many decision nodes to go through before each model could arrive at a result. Because of this, even a small change in input variable could result in completely different regression models, hence leading to the phenomenon that we observed.

The overall performance of the Lasso regression on our dataset was fairly good for articles from Moreover, but not as satisfactory for articles from Opoint. The scores obtained after a 10-fold cross validation and hyper-parameter tuning on alpha did not lead to remarkable improvement. In general, we expected that hyper-parameter tuning on alpha would enhance the performance of the model, since using smaller alpha values reduces the amount of penalization (Jain 2017); however, the results obtained after cross validation proved

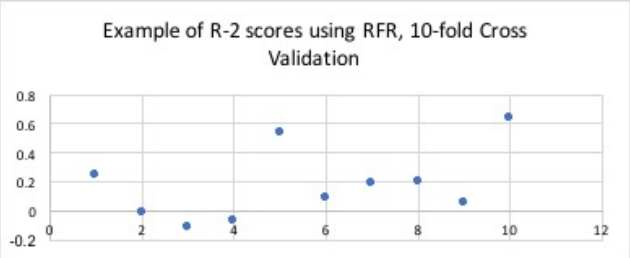

Figure 2: Results of R-2 scores derived from the RFR model, using n_estimator = 100, max_feature = 'sqrt' and max_depth = 20.

that Lasso performed better than `RFR` on the Moreover articles, and it performed significantly worse on the articles from Opoint.

As mentioned in the previous sections, later on in our project we realized that without deleting duplicates in our data frame, we were essentially overestimating our model performance, since we were testing the model on what it was trained on. The results we delivered in Figure 1 were recorded after we had deleted the duplicates. We compared the two sets of results before and after we discarded duplicates, and the results are shown in Figure 3. Interestingly, `RFR` was affected a lot more than `LinearSVR` was, perhaps due to the fact that `LinearSVR` is well-performed for text analytics problem to begin with. However, keeping duplicates simply would create excessive optimism for `RFR`'s model performances.

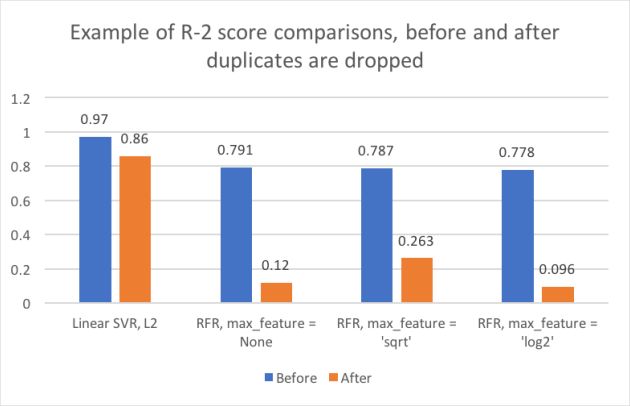

Figure 3: Visualization of the effects after duplicates were dropped

5 Conclusions

In this paper, we set out to build a model that could accurately predict the popularity of news articles in terms of momentum scores. After converting text data in to trainable,numerical values, we built and trained our models, and discovered that `LinearSVR` came away with the best results.Our most important discovery is that increasing the number of trees in the `RandomForestRegressor` model

mitigated the high variance problem we saw in our previous results by a fair amount. Perhaps using a large number of trees is required for this problem, given the immense feature size we had.

Should we have more time in the future, we would like to go a few steps further with pre-processing our data. First of all, we would like to add a few meta data properties to be the feature data. Recall that for this project, we only considered the text content of individual articles as our feature. However, meta data such as the specific website where an article is published, its author, or even the time it was published can potentially affect its popularity too. Secondly, we would like to make sure all duplicates are deleted for a less biased result. The `panda`'s drop_duplicates function allowed us to throw away articles that have the same titles. However, we found out later in our project that this approach didn't do the entire job as some articles had the exact same contents and momentum scores, but had different titles. Lastly, we would trim down our data set even more and take away more words that do not contribute to the popularity measure. For this project, we only took out common English words like "a, the, it", etc. However, we believe that only a small number of words should matter for the popularity measure, and sometimes only one keyword such as *Donald* could be responsible for a high score.

6 Contributions

Huseyin and Sophie worked on the pre-processing of the data and wrote the code for `LinearSVR` and `RandomForestRegressor`, and `Lasso Regressor`. For the paper, Sophie wrote the Abstract, Introduction, and Background. Both partners wrote the Experiment, Results, and the Conclusion sections together. Both partners proof-read before submitting the paper.

7 Acknowledgements

We would like to thank Quoin, the Charlotte based data science company for providing the dataset for our project, as well as Dr. Ramanujan for helping us interpret the datasets. We would also like to thank our friends and classmates for their generous encouragement and support throughout the project.

References

Albon, C. 2017. Effect of alpha on lasso regression. `https://chrisalbon.com/machine_learning/linear_regression/effect_of_alpha_on_lasso_regression/`. Retrieved on Apr. 10, 2018.

Jain, S. 2017. A comprehensive beginners guide for linear, ridge and lasso regression. `https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/`. Retrieved on Apr. 10, 2018.

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