Online News Popularity: A Regression to Predict Views

Haley Farber, Alice Hua, Derrick Xiong

Introduction

Over the years, online media has taken over as the main source of information in place of the traditional, paper-based media. In 2020, we saw the number of people around the world using the internet grew to 4.54 billion, an increase of 7 percent (298 million new users) compared to January 2019. At the same time, we are showered with an unprecedented amount of information from all the platforms. Therefore, predicting whether a piece of an article will receive popularity is of vital importance in this digital age. In this study, we aimed to use regression to analyze and predict the views of Forbes articles. Although many studies have been conducted on datasets like Mashable, we decided to build up our own dataset from the ground up and engineered all the important features using LDA and keyword extraction. We hope that, at the end of the study, we have provided a solid model that performs reasonably well and a robust pipeline of data collection and feature engineering that can be easily generalized to collect more data from other sources for researchers still yet to come.

Literature Review

Much of the research done on online news popularity stems from the article, "A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News" by Kelwin Fernandes , Pedro Vinagre , and Paulo Cortez. They were the creators of the "Online News Popularity Data Set" which they donated to the UCI Machine Learning Repository. They collected data from 2013 - 2015. We reviewed a paper by He Ren and Quan Yang, written in 2015 that uses the Mashable dataset and predicts the number of shares a Mashable article has through multiple models including SVM, Random Forest, k-Nearest Neighbors, and Logistic Regression. They also ran an indirect application of linear regression due to the high variance of shares. The next paper we read was by Choudhary & Sandhu & Pradhan in 2017. Instead of predicting the number of shares, they binarized the number of shares to be popular or not popular based on the average number of shares. They ran correlation analysis to select only the positively correlated features from the Mashable dataset which greatly improved their prediction accuracy. What made their paper interesting was the binarization of the number of shares to a classification of popularity. We were also intrigued by their method of correlation analysis which we ended up using in our own models later. The final paper we reviewed was by Joe Johnson and Noem Weinberger for their computer science class, cs229 at Stanford University. They too ran classification based on binarizing shares on the Mashable; however, unlike the first two papers, they decided to run a direct linear regression using the number of shares. They used methods such as ridge and lasso regression to decrease overfitting.

While most people have used the Mashable dataset for a few years beyond its release, we felt that the results from these articles may not be accurate for more recent articles. It is currently 2020 and the last of the data was gathered in 2015; given how fast online platforms change and what people are interested in as well, we decided to gather gather fresh data from 2020. Furthermore, 2020 is an incredibly unusual year due to the widespread coronavirus so articles that are popular this year and perhaps in the near future may not reflect what would have been popular in 2015. We also decided that we, like Johnson and Weinberger, want to predict the continuity of popularity (i.e. through the number of shares or views an article has) versus simply predicting if an article will be popular or not. Thus, we decided to collect data from 2020, recreate the features from the Mashable dataset, and run a linear regression on these features.

Data

Data Scraping

We started to collect data from Mashable. Mashable has a standard HTML structure for most of its articles and has an archive for each month and year so we were able to collect data by selecting the month we wished to extract data for. However, we soon discovered that Mashable changed their website to not include the number of shares per article so we could not extract this information from HTML.So, we decided to use Forbes, another online news site, to gather our data instead. Forbes includes the number of views an article has in the headline above most of its articles so we decided to use views instead of shares to measure the popularity of an article. However, Forbes unlike Mashable has a different HTML structure for most of its articles so we could not simply search for one class containing the article content. Thus, we had to create a different approach than the simpler one we used to initially web scrape Mashable and look for multiple, different classes that could contain an article's content. Another problem we encountered was that Forbes did not have an archive so we could not look for articles by the date they were posted. Instead, we scraped for articles by their topic and scraped the date of the article from HTML. One final problem we encountered when we began to scrape large amounts of data from Forbes was their pay-wall; we could not get access to more articles without paying for a subscription. Thus, we paid for a subscription and continued to scrape until we finally gathered around 7k data points. We would like to have scraped more than 7k data points, but could not given the time constraints and how difficult it was to scrape data from Forbes. However, we felt that 7k data was enough to begin our research and use to make generalizable conclusions about online news popularity. After we extracted about 7k worth of data from Forbes, we began to create the features.

Feature Engineering

Most of these features were built and named after the features used by Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez from their dataset that was used to measure the popularity of Mashable articles from 2013-2015. We ended up using 52 out of 60 of their features to construct our own features from the Forbes data. The only ones we did not include were attributes that were not included in the Forbes HTML or were too time-consuming for us to create new functions to parse through the different HTML structures to web scrape. These attributes are the following: number of images, number of videos, number of links, number of links to other Forbes articles, Min. shares of referenced articles in Forbes, Max. shares of referenced articles in Forbes, Avg. shares of referenced articles in Forbes. We added 12 additional dummy variables to our features account for the month an article was posted in and a column for stating which month the article was posted for for ease of use in calling the months for our models later. We encountered a few issues recreating the features from the original Mashable dataset and learned a lot about feature engineering in the process.

The first issue we encountered was that the original article by Fernandes, Vinagre, and Cortez did not include any source code or pseudo code for they derived their features so we had to take some creative license and make our best guess as to how they derived these features. Their paper had this chart though which offered more detail than the UCI learning repository with the feature names listed.

Table 2: List of attributes by category.

Feature	Type (#)
Words	
Number of words in the title	number (1)
Number of words in the article	number (1)
Average word length	number (1)
Rate of non-stop words	ratio (1)
Rate of unique words	ratio (1)
Rate of unique non-stop words	ratio (1)
Links	
Number of links	number (1)
Number of Mashable article links	number (1)
Minimum, average and maximum number of shares of Mashable links	number (3)
Digital Media	
Number of images	number (1)
Number of videos	number (1)
Time	
Day of the week	nominal (1)
Published on a weekend?	bool (1)

Feature	Type (#)
Keywords	
Number of keywords	number (1)
Worst keyword (min./avg./max. shares)	number (3)
Average keyword (min./avg./max. shares)	number (3)
Best keyword (min./avg./max. shares)	number (3)
Article category (Mashable data channel)	nominal (1)
Natural Language Processing	
Closeness to top 5 LDA topics	ratio (5)
Title subjectivity	ratio (1)
Article text subjectivity score and its absolute difference to 0.5	ratio (2)
Title sentiment polarity	ratio (1)
Rate of positive and negative words	ratio (2)
Pos. words rate among non-neutral words	ratio (1)
Neg. words rate among non-neutral words	ratio (1)
Polarity of positive words (min./avg./max.)	ratio (3)
Polarity of negative words (min./avg./max.)	ratio (3)
Article text polarity score and its absolute difference to 0.5	ratio (2)

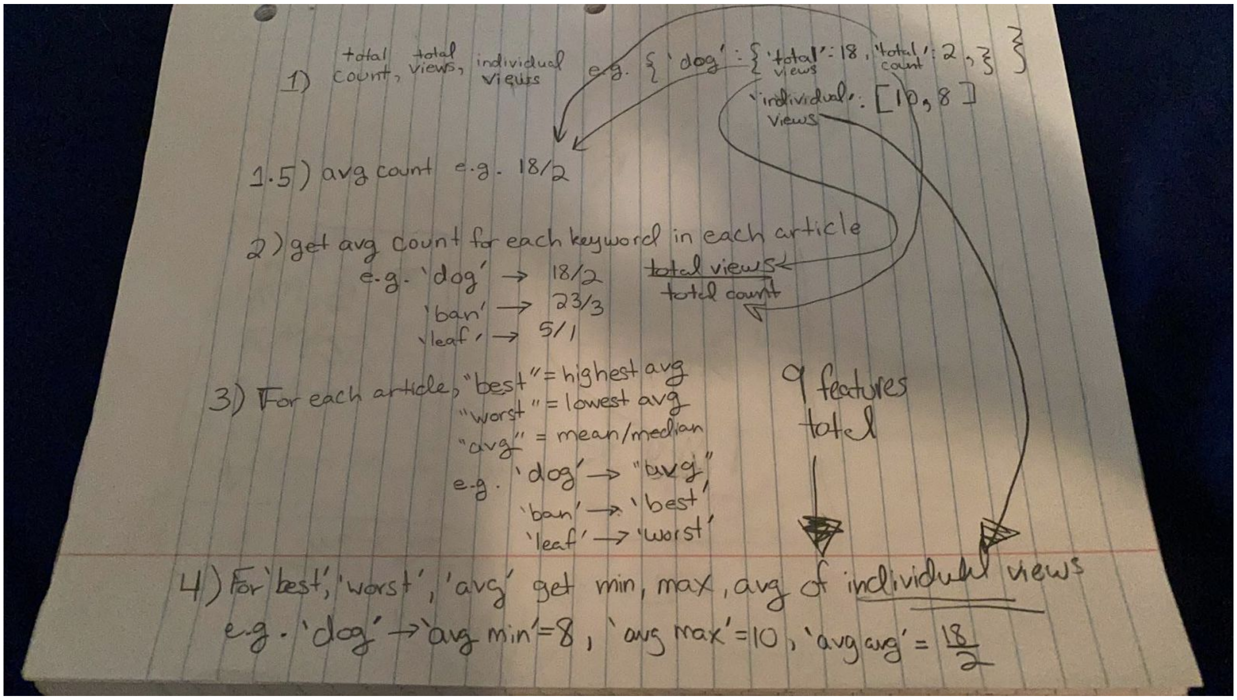
Target	Type (#)
Number of article Mashable shares	number (1)

We combined the more thorough descriptions of the variables found here with the actual feature names from the UCI learning repository. The first issue we ran into when recreating these features was the package to use to recreate the subjectivity and polarity features. The authors of

the original Mashable dataset said they used the Pattern web mining module (http://www.clips.ua.ac.be/pattern), but when we clicked this link, we discovered that the page no longer exists. Thus, we decided to research the Pattern web mining module on our own. However, we discovered little documentation on this module and decided to instead use the popular and more documented TextBlob to create our subjectivity and polarity features. Another small issue we discovered was that the authors put the incorrect description for their abs_title_sentiment_polarity feature. The author's chart says that this variable is calculated by taking the absolute difference of an article's text polarity score to 0.5; however, it is calculated in the csv of their data by only taking the absolute value. We corrected for this by only taking the absolute value for this feature as they did in the csv and not taking the difference to 0.5 as they did in both the chart and csv for their other feature, abs_title_subjectivity. Another small difference was that the chart says that they find the abs value of the article text subjectivity score and its absolute difference to 0.5, but the features dictionary on the UCI learning repository only includes absolute value features for an article's title. Thus, we only performed these absolute operations on an article's title.
One particular feature we struggled to understand and follow were the 9 keyword features:

1. kw_min_min: Worst keyword (min. shares)
2. kw_max_min: Worst keyword (max. shares)
3. kw_avg_min: Worst keyword (avg. shares)
4. kw_min_max: Best keyword (min. shares)
5. kw_max_max: Best keyword (max. shares)
6. kw_avg_max: Best keyword (avg. shares)
7. kw_min_avg: Avg. keyword (min. shares)
8. kw_max_avg: Avg. keyword (max. shares)
9. kw_avg_avg: Avg. keyword (avg. shares)

The authors of the original dataset did not leave much description for how they discovered these features or what they meant. One initial difference we noticed was that the authors were able to retrieve the keywords from the article's metadata and then calculate these features although they did not specify exactly how they did this. We did not have HTML data for an article's keywords so we discovered an article's keywords using the gensim package. We then looked online to see if anyone had discovered how to replicate these features or if these were commonly used features in NLP. We could not completely understand the concept by searching online so we reached out to a TA with much more experience in NLP than us, Gurdit Chahal, who was kind enough to draw us the diagram below and walk us through what he thought the features meant.



This explanation matched the more vague descriptions of the keyword features and the values we received after running it were in the same scale as those in the original Mashable dataset so we believed this to be correct and ran it as such. What we learned about the nine keyword features is that you first must find all the keywords in an article using the gensim package and save them in a list. Then, for every keyword, you must find all of the articles that contain this keyword, record the number of articles that contain the keyword and record a list of the number of views each of these articles contains. You must also then store the value of the sum of these counts. Once you have calculated the total counts, individual views, and total views for each keyword you must calculate the count that the feature specifies. You then calculate the average count for each keyword in each article by dividing the total number of views by the count of articles and store this in a dictionary. Each keyword feature has two components from avg, min, max; for example, one keyword feature is kw_avg_max. The middle term represents the type of keyword you are looking for. Avg means you are finding the keyword for each article that has the mean of the average count per keyword. Min means you are finding the "worst" keyword or the keyword with the lowest average count per keyword. Max means you are finding the "best" keyword or the keyword with the highest average count per keyword. The second term represents the type of count you are looking for after you have decided which type of keyword to use ("avg", "worst", "best"). Avg means you must find the average count for the keyword you specified in the second term and return its number of views. Min means you must find the article with the least number of individual views from the keyword you specified in the second term and return its number of views. Max means you must find the article with largest number of views from the keyword you specified in the second term and return its number of views. This is how we calculated the nine keyword features.

Another set of features that was challenging for us to calculate were the LDA features. The original authors of the Mashable dataset stated that they used the Laten Dirichlet Algorithm to identify the relevant topics and then measure the closeness of each article to these topics. Thus, we did some research on the Latent Dirichlet Algorithm since this was a topic none of us were familiar with. We used source code from a few “towards data science” articles, all of which are cited in the Features notebook included in our repo. The goal of LDA is to determine the topic an article belongs to by discovering which words belong to documents and what words belong to certain topics. The first step for this was preprocessing our data by lowercasing and tokenizing words through gensim.utils.simple_preprocess and removing stop words and only keeping words greater than three characters and then lemmatizating and stemming the words. The second step was to create a bag of words on the data set. We first created a master dictionary for all of the words found in the entire data set. We indexed the words so we could use these indices in the individual dictionaries. We created an individual dictionary for each document reporting how many many times each word showed up in the document. We then ran LDA using the gensim package and bag of words. We decided to set the number of topics to five since we are using five topics - the authors of the original Mashable dataset had six topics but decided to use LDA to select the most five relevant topics as well. Running LDA yields the list with the probabilities for each topic per document with index for topic. We then extracted these probabilities for each topic per document and put them into our dataframe as LDA features. Thus, LDA and keywords were the features that took us the most time as a team to learn and understand. The code for the other features can be found in the Features notebook in our repo. Here is a dictionary of attributes we scraped from Forbes to create our features and a dictionary of our features:

What is Given From Forbes Scraping:

1. link: URL of the article (non-predictive)
2. title: title of the article
3. text: article content
4. views: number of views in text
5. topic: given subtopics from Forbes
6. time: time article was published

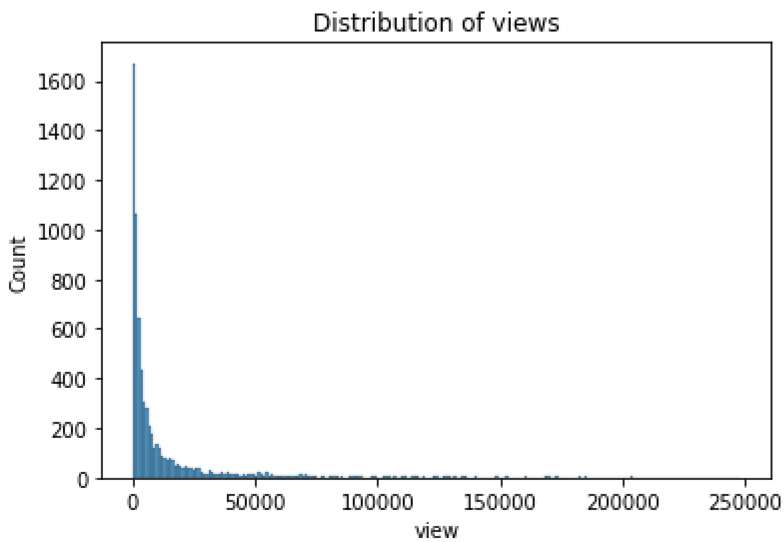
Features Added:

10. Innovation: dummy variable for articles in the innovation topic
11. Leadership: dummy variable for articles in leadership topic
12. Lifestyle: dummy variable for article in lifestyle topic
13. Money: dummy variable for article in money topic
14. month: month article was published
15. Month dummies (11 total) - Jan, Feb, Mar, Apr, May, Jun, July, Aug, Sep, Oct, Nov
16. n_tokens_title: Number of words in the title
17. n_tokens_content: Number of words in the article
18. n_unique_tokens: Percent of unique words in the article

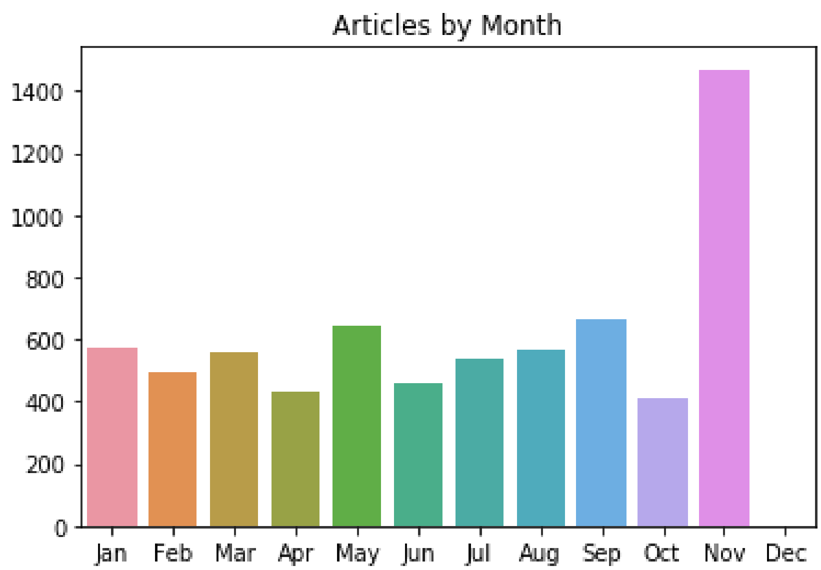
19. average_token_length: Average length of the words in the content
20. n_non_stop_words: Percent of non-stop words in the article
21. n_non_stop_unique_tokens: Percent of unique non-stop words in the article
22. day_of_week: day of the week the article was published
23. Day of week dummies (7 total) - monday, tuesday, wednesday, thursday, friday, saturday, sunday
24. Weekend_or_weekday: was article published on weekend or weekday
25. weekday: dummy variable if article was published on weekday
26. weekend: dummy variable if article was published on weekend
27. global_sentiment_polarity: article sentiment polarity
28. global_subjectivity: subjectivity of article conten
29. abs_title_sentiment_polarity: Absolute polarity level
30. title_subjectivity: Title subjectivity
31. abs_title_subjectivity: Absolute difference of title subjectivity level - 0.5
32. title_sentiment_polarity: Title polarity
33. global_rate_positive_words: Rate of positive words in the article
34. global_rate_negative_words: Rate of negative words in the article
35. rate_positive_words: Rate of positive words among non-neutral tokens
36. rate_negative_words: Rate of negative words among non-neutral tokens
37. avg_positive_polarity: Avg. polarity of positive words in an article
38. min_positive_polarity: Min. polarity of positive words in an article
39. max_positive_polarity: Max. polarity of positive words in an article
40. avg_negative_polarity: Avg. polarity of negative words in an article
41. min_negative_polarity: Min. polarity of negative words in an article
42. max_negative_polarity: Max. polarity of negative words in an article
43. LDA_00: Closeness to LDA topic 0
44. LDA_01: Closeness to LDA topic 1
45. LDA_02: Closeness to LDA topic 2
46. LDA_03: Closeness to LDA topic 3
47. LDA_04: Closeness to LDA topic 4
48. kw_min_min: Worst keyword (min. shares)
49. kw_max_min: Worst keyword (max. shares)
50. kw_avg_min: Worst keyword (avg. shares)
51. kw_min_max: Best keyword (min. shares)
52. kw_max_max: Best keyword (max. shares)
53. kw_avg_max: Best keyword (avg. shares)
54. kw_min_avg: Avg. keyword (min. shares)
55. kw_max_avg: Avg. keyword (max. shares)
56. kw_avg_avg: Avg. keyword (avg. shares)
57. timedelta: Days between the article publication and the dataset acquisition (non-predictive)
58. num_keywords: Number of keywords in the article

EDA

Once we extracted all the features. We did a quick EDA on the data set and examined the different features and outliers. The most obvious problem is the distribution of views. The number of views is not normally distributed and has a lot of outliers, which makes regression difficult. The article that has the highest number of views is the one titled "20-Year-Old Robinhood Customer Dies By Suicide After Seeing A $730,000 Negative Balance". This article has more than 18 million views while the 75 percentile of the views of our entire dataset is 12,380. Data points like this make the distribution of our outcome variable very skewed. We addressed this problem by eliminating such outliers. We dropped the articles that have views more than 250000, which accounts for 2% of the total data. And then we dropped the articles that have views less than 200, which accounts for roughly 1.5% of the total data. After the elimination of outliers, the distribution is still heavily skewed as shown below but the effect with extreme outliers is significantly reduced.



The second problem we have from EDA is the unequal distribution of data points across different months. Since we started data collection by gathering data from November, a large portion of the data is in November compared to other months as shown below. This will impact the performance of our model but it should be a very simple problem to fix once we gather more data and make sure they are evenly distributed between the months.



Models and Results

We started a model building process and implemented the OLS model as well as the training scheme. Our plan is to use the rolling window technique which separates the data points into 11 months and uses different windows of past months to predict the views of 1 future month. For example, we started training OLS models on data from January, February, and March to predict the views from articles from April. Our metric is percentage error, which is measured by calculating the percentage between the predicted view and true view of that article. The results of this model, however, were terrible as shown below.

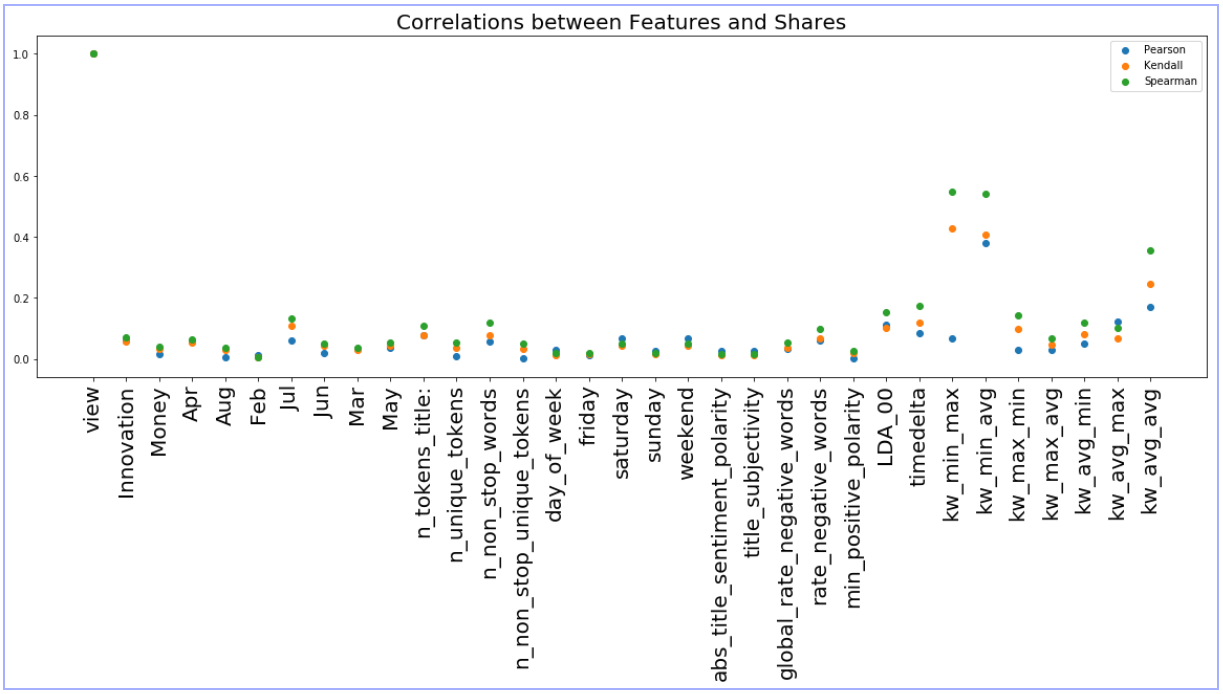
```
In [333]: pred
```

Out[333]:

	true_view	predict_view	pct_error
2	4064	-6.719910e+08	-1.653531e+07
22	3540	-6.720025e+08	-1.898322e+07
82	1645	-6.719964e+08	-4.085095e+07
114	662	-6.720035e+08	-1.015112e+08
148	1684	-6.719961e+08	-3.990486e+07
...	...	...	...
5694	512	-6.719787e+08	-1.312459e+08
5695	3152	-6.719843e+08	-2.131940e+07
5697	3097	-6.719789e+08	-2.169784e+07
5720	6817	-6.719607e+08	-9.857232e+06
5738	2054	-6.719736e+08	-3.271547e+07

408 rows × 3 columns

Then we noticed that the issue lied in the huge number of features we have. Out of the 68 features we managed to extract from the texts, most of them had little correlation to views and only contributed to adding noise to our model. So we decided to trim down the number of features by finding the ones that have the most correlation with our outcome variable: view. We calculated 3 different correlations: Pearson, Kendall' tau, Spearman's rank correlation and chose the features that have positive correlations with the views. The results are as follows.



From this point on, we started training our OLS model only with features that are positively correlated with views. And we got a significant increase in our predicting accuracy.

```
In [345]: pred
Out[345]:
```

	true_view	predict_view	pct_error
2	4064	2268.734393	-44.174843
22	3540	1361.347454	-61.543857
82	1645	-4532.256416	-375.517107
114	662	-4162.322176	-728.749573
148	1684	-707.909079	-142.037356
...	...	...	...
5694	512	17320.722842	3282.953680
5695	3152	14174.035234	349.683859
5697	3097	17563.426713	467.110969
5720	6817	33285.688613	388.274734
5738	2054	25485.580065	1140.777997

408 rows × 3 columns

Then we implemented the rolling window of 3 months to train 7 different models across the year 2020. Here we introduced a new metric, average percentage error, which measures the average of the percentage errors across all the predictions of that given month. We get the results as follows.

```
In [353]: result
Out[353]:
```

	Train_month	Test_month	Average_pct_error
0	Jan, Feb, Mar	Apr	269.543188
1	Feb, Mar, Apr	May	380.486238
2	Mar, Apr, May	Jun	1193.665884
3	Apr, May, Jun	Jul	1125.715705
4	May, Jun, Jul	Aug	1713.486585
5	Jun, Jul, Aug	Sep	1587.295163
6	Jul, Aug, Sep	Oct	70.649050
7	Aug, Sep, Oct	Nov	316.077906

From this point on, we noticed the average percentage error is still very high, although significantly better than before. Then we started to think that the still relatively high percentage errors were caused by overfitting and the still ever-present outlier problems so we introduced two modifications to our regression. The first one is the L2 regularized OLS or ridge regression, while the second one is the Random sample consensus (RANSAC). We incorporated these two models and tried to compare the results between the 3 models we had so far and got the following results:

Out[354]:

	Train_month	Test_month	Average_pct_error_OLS	Average_pct_error_Ridge	Average_pct_error_RANSAC
0	Jan, Feb, Mar	Apr	319.729076	356.004239	1392.898791
1	Feb, Mar, Apr	May	401.005362	368.403006	398.307972
2	Mar, Apr, May	Jun	1193.665884	298.099140	217.026743
3	Apr, May, Jun	Jul	1125.715705	242.256164	396.768366
4	May, Jun, Jul	Aug	1713.486585	347.450707	171.833617
5	Jun, Jul, Aug	Sep	1587.295163	347.152479	107.398336
6	Jul, Aug, Sep	Oct	167.605017	127.312918	1353.532792
7	Aug, Sep, Oct	Nov	354.960497	305.893875	599.146119

Here we can see the error percentage of ridge regression is very consistent across the 7 different models which is to be expected from regularization but the percentage error across the 3 different models is still higher than what we were expecting.

We then realized that although the new models do show promising results, the problem still lies in the fact that we have too many features that don't contribute significantly to the prediction of views. If we look at the graph that plots the correlation of the variables that are positively correlated with the number of views, we notice most variables have very small correlations. Therefore, we decided to fine-tune the different features and experiment with different combinations and find the ones with the lowest percentage error. Additionally, we decided to expand the window from 3 months of training to 6 months of training, thereby increasing the performance of our model by training on more data. The final results we got were very promising and showed great improvement from our initial model. They are as follows:

	Train_month	Test_month	Average_pct_error_OLS	Average_pct_error_Ridge	Average_pct_error_RANSAC
0	Jan, Feb, Mar	Apr	477.185388	466.504964	62.594194
1	Feb, Mar, Apr	May	487.871928	454.136389	163.364450
2	Mar, Apr, May	Jun	461.134148	422.646195	20.804344
3	Apr, May, Jun	Jul	374.704348	347.975590	31.416039
4	May, Jun, Jul	Aug	439.161979	415.720109	183.568721
5	Jun, Jul, Aug	Sep	696.576601	692.576029	246.302175
6	Jul, Aug, Sep	Oct	500.632742	497.333328	66.804967
7	Aug, Sep, Oct	Nov	515.204290	535.530223	134.833123

	Train_month	Test_month	Average_pct_error_OLS	Average_pct_error_Ridge	Average_pct_error_RANSAC
0	Jan, Feb, Mar, Apr, May, Jun	Jul	357.296020	340.065529	12.765784
1	Feb, Mar, Apr, May, Jun, Jul	Aug	409.791899	386.235687	14.712310
2	Mar, Apr, May, Jun, Jul, Aug	Sep	666.568385	634.645327	89.812642
3	Apr, May, Jun, Jul, Aug, Sep	Oct	523.779937	498.377277	47.477806
4	May, Jun, Jul, Aug, Sep, Oct	Nov	765.292144	727.014411	94.771695

Conclusion

In conclusion, we realize, after many iterations of models, that given the distribution of our data, special care needs to be taken in handling the outliers and high variance in our data. Our regression model provides a good baseline for the prediction of views, but better models such as neural networks could be implemented with more data to achieve better results. Our research explored the possibility of using different regression models to predict views, which is an essential metric for popularity. Although there have been many studies done with the available Mashable dataset, our quest started with creating a new dataset from the ground up and focused on using regression to tackle a more challenging task of predicting the view as a continuous variable as opposed to a binary (popular/non-popular) variable. During the process, we have learned a great deal about data collection, feature engineering, and model selections. And in the end, we have achieved promising results with the limited time and data we have. Last but not the least, our data collection and feature engineering process could be generalized to collect more data from many different sources other than Forbes to create more diverse and robust datasets for future research endeavors beyond the prediction of views and popularity.

Works Cited

Choudhary, Swati & Sandhu, Angkirat & Pradhan, Tribikram. (2017). Genetic Algorithm Based Correlation Enhanced Prediction of Online News Popularity. 10.1007/978-981-10-3874-7_13.

Fernandes, Kelwin, et al. “A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News.” *SpringerLink*, Springer, Cham, 25 Aug. 2015, link.springer.com/chapter/10.1007/978-3-319-23485-4_53.

Kemp, Simon. “Digital Trends 2020: Every Single Stat You Need to Know about the Internet.” *Growth Quarters | The Next Web*, 6 Feb. 2020, thenextweb.com/growth-quarters/2020/01/30/digital-trends-2020-every-single-stat-you-need-to-know-about-the-internet/.

Ren, H., and Quan Yang. “Predicting and Evaluating the Popularity of Online News: Semantic Scholar.” *Semantic Scholar*, 2015, www.semanticscholar.org/paper/Predicting-and-Evaluating-the-Popularity-of-Online-Ren-Yang/120164b6de44b23f5bf3ffc91f38c64313486372.

Johnson, Joe and Noam Weinberger. Predicting News Sharing on Social Media. PDF file. 2016. http://cs229.stanford.edu/proj2016/report/JohnsonWeinberger-PredictingNewsSharing-report.pdf