What features did you consider?

First, I considered the features that were already provided to us through shared_articles.csv. However, I realized a regression model would not be able to use categorical data given to us for authorPersonId, authorRegion, authorCountry, url, lang, timestamp, and content in its raw form. Therefore, I decided to use one-hot encoding to represent the different values as feature columns in my dataframe. I also ignored authorSessionId because the categorical data would not impact the virality score as it is assigned to an author every time they login.

The one-hot encoding approach seemed to work well with authorPersonId, authorRegion, authorCountry, contentType, and lang, as all of them had a reasonable number of unique values.

I stripped the urls to their domains (http://www.nytimes.com/2016/03/28/busine... -> http://www.nytimes.com/) in an attempt to create categorical data so that I could one-hot encode it. However, there were more than 1000 unique domains that the articles belonged to, so it was not a discriminating enough feature to help our model. Not surprisingly, the including this feature worsened the model's performance.

For timestamp, I thought the day of week that the article was posted could help decide the virality score, so I converted timestamp to datetime and then extracted the day of week. I then one-hot encoded this so that it would work with our regression model. It turned out to be a very helpful feature.

Another feature I decided to experiment with was text length which I extracted from the text - how long the article is may have an impact on its virality. This turned out to be the most important feature in our model according to random forest regression feature importances.

I also wanted to extract information from the title and text, because the content of those data points would definitely have an impact on the article's virality. I explored different techniques to do this, and tried to extract the top keywords that appear in the title and text of every article. However after extracting this information, I realized that it would be difficult to present it in a way that a model would understand. Therefore, I decided to use a one-hot encoding of the top 10 keywords that appear in articles with a high virality score (>50). For each article, I would simply encode whether each of the top keywords appeared in its title. The keywords were generated by selecting the words with the tf-idf scores across all the titles.

What model did you use and why?

I used a Random Forest Regression model to predict the virality scores, based on the features I had generated. The reason for choosing a regression model over a classification model is that I wanted to predict a discrete value for virality. One of the reasons for choosing a random forest regression is that it is able to very easily show which of my features were most important. While a linear regression model could also show this, it would not be as interpretable.
While comparing between linear and random forest regression, I chose RF regression

because linear regression is known to not do so well with many columns created by one hot encoding. This could be because of the potential multicollinearity caused by onehot encoding that negatively impacts linear regression. Random forest regression deals with this better.

What was your evaluation metric for this?

My evaluation metrics for this were the MSE (Mean Squared Error), the R^2 score (Coefficient of Determination) and the MAE (Mean Absolute Error). The MSE helped identify how close I was to the actual value of each virality score, and the R^2 score helped show how well the model fit to the dependent variables. The MAE can prove useful because it treats all errors the same instead of penalizing larger errors like MSE. I experimented by dropping / adding different features and testing a random forest regression model and a linear regression model.

The final metrics for the random forest regression model were:
MSE: 7809.38
R^2: 0.82
MAE: 48.37

What features would you like to add to the model in the future if you had more time?

One important feature would be the time of day that the article was published. This could be useful because articles shared in the morning may be more likely to be more viral. This information would have to be binned into either 24 hours in the day or some other categorical form (morning, afternoon, ..), and then one-hot encoded. One catch with this is that different regions / countries have different time zones. So this timezone information should be accounted for consistency.

Another feature I would have liked to incorporate was url_virality_percentile. I would sort the URLs by their average virality score and then convert it into a percentile for our model. This would allow the model to use the website where the article was posted to predict how viral the article would be. Similarly, I would do this for authorId as well.

Just like I did for titles, I would have also liked to extract the top keywords for texts and create a one-hot encoding of that to see if that would help train a better model.

What other things would you want to try before deploying this model in production?

Using the percentile system for URLs and authorIds would mean we would have to account for websites and authorIds that we don't have

previous information on or are simply new. I would set up mechanisms to handle edge cases like this before productionizing the model.

I would also like to join our data with more external data to receive information like the number or users / popularity of each blog post. I would then set up a database to store this information as well to decrease runtime and memory usage.

Specifically for the model, I would try a gridsearch to find optimal model parameters, and conduct K-fold cross validation to ensure that the model is not overfitting.

Perhaps the most important thing I would like to try is to establish a trend score, to identify the articles with keywords that are becoming increasingly or decreasingly popular with time. I would do much more research about NLP and time series techniques to see how this could be done.