Machine Learning Engineer Nanodegree

Capstone Project

Predicting Online News Article Popularity

Thomas Kauth
July 14, 2018

Definition

Project Overview

The rise of the internet has shaken up the world of print journalism. Traditional news sources have all been forced to undergo some degree of transformation, and many are still struggling with relevance and profitability in the 21st century. Across the sphere of the internet, they must compete with dozens of non-traditional news sources.

One relevant question that all news sources should be asking is, what makes a news story popular? Analyzing data about what qualities appeal to readers can lead to simple editorial improvements -- e.g., the length of the title, length of the content, positive/negative words in the content, etc. -- that may drive popularity more than writers and editors realize.

This project will attempt to determine the most important qualities that drive news popularity. The dataset for this project is posted at the UCI Machine Learning Repository. The news articles were all posted to the website Mashable during a two year period ending in 2015, but the original content is not part of the dataset. Instead, researchers Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez[1] analyzed the content to create the dataset.

Problem Statement

To gain insight into why a news article can achieve popularity, we need a target measurement. Page views are one metric, but it is impossible to know if the reader read the article and liked it, read it and hated it, or read the first paragraph and then gave up. A better metric is user shares. If a user is sharing the article, that is a very good indication that that user enjoyed the article enough to share it with others.

Additionally, we also need metrics describing the content of each article in objective terms. If we are to answer the question of why one article is more popular than another, we need a way to measure the different qualities of the articles – i.e., we need a number of objective descriptors of the contents of the articles in the dataset.

With good descriptive categories and a target measurement, we can then apply a number of classification machine learning techniques to find the best predictor of news article popularity, and then determine which of the descriptive categories are most relevant for predicting popularity.

---

1 K. Fernandes, P. Vinagre and P. Cortez. A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News. Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence, September, Coimbra, Portugal.

This problem, then, is a single class supervised classification problem. The input data (X) is the 58 predictive features provided by the dataset (e.g., number of words in the article title, or number of words in the content) and the output (y) is a prediction of high popularity or low popularity, with popularity defined as greater than the median number of shares.

The dataset will be divided into 50% training data, 25% validation data, and 25% testing data. I'll train the various classifiers on the training data, and use GridSearch to test the various classifier hyperparameters, utilizing the validation data as the data source. Once I've found the best classifier/hyperparameter combination for predicting popularity against the validation data, I'll discover the most relevant descriptive features by using that classifier's feature importances function. This will then give me a list of the top ten most important features for determining popularity.

The dataset provides both the `shares` target value and 58 objective descriptors of each article's contents. To predict popularity I will try the following classification algorithms to determine which one does the best job of classifying against the validation data:

- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Random Forest
- AdaBoost
- XGBoost
- Naive Bayes

I'll also try various neural network configurations to see if I can achieve better prediction results using a neural net. I expect the solution will likely be Random Forest (which is what Fernandes, et al. discovered to be best), or one of the gradient boosting algorithms – AdaBoost or XGBoost. I've found boosting algorithms to work quite well on classification problems in the past.

Metrics

I plan to use the same evaluation metrics that Fernandes, Vinagre and Cortez used in their study[2] – Accuracy, Precision, Recall, F1 score and Area Under Curve of the Receiver Operating Characteristic (AUC of the ROC).

Accuracy

Accuracy is simply `Correct Predictions / All Predictions`

Precision

Precision is `True Positives / (True Positives + False Positives)`

Recall

Recall is `True Positives / (True Positives + False Negatives)`

2 Ibid., 8

F1

F1 combines Recall and Precision:
`2 * (Precision * Recall) / (Precision + Recall)`

AUC of the ROC

AUC of the ROC is the area under the plot of `True Positive Rate / False Positive Rate` across varying classification thresholds. True Positive Rate is equal to Recall above. False Positive Rate is its opposite: `False Positives / (False Positives + True Negatives)`

The Area Under the Curve of the Receiver Operating Characteristic is not as intuitive as the other metrics above. It determines accuracy by varying the threshold at which a probability turns from 0 to 1. For example, if many of the probabilities returned by an algorithm hover near 0.5 (e.g., 0.501, 0.503, 0.497, 0.499), then a high accuracy score could be misleading. With just a minor perturbation of the data, many of the probabilities could slip from 0 to 1 or from 1 to 0 because they are so close to the 0.5 mark.

A higher AUC of the ROC implies higher confidence because the predicted probabilities of the 1 values are significantly higher than 0.5 and the predicted probabilities of the 0 values are significantly lower than 0.5 than a lower AUC of the ROC score. Hence a higher AUC of the ROC implies a more robust model. If there is a perturbation to the data, the model should still do a good job of classifying.

To my mind, Accuracy and AUC of the ROC are the two metrics to observe in this study. I'll be focusing a bit more on AUC of the ROC because it implies both an accurate and a robust model, and it is also the metric on which Fernandes, et al. focused.

Recall, Precision and F1 score are great metrics for unbalanced datasets – where a high percentage of the data should be classified as a 0 and a small number should be classified as a 1, for example. But this dataset is balanced by definition – it is divided in half, with approximately half of the articles labeled popular and half unpopular. The reason I'm including Recall, Precision, and F1 here is purely for comparison to the outcomes reported by Fernandes, et al.

And like Fernandes, et al., I plan on undertaking a study of feature importance, a metric that has various implementations, but which attempts to uncover the importance of a given feature by measuring the difference in error when the feature is removed or via other similar methods. In the Fernandes study Random Forest was found to be the most accurate classifier, and they used the `feature_importances` method of the Random Forest classifier. I plan to do the same type of study, depending on which classifier returns the most accurate predictions. I will then compare my rankings to theirs. The results should suggest insights into what features a news writer or editor could modify to garner more interest from readers and increase readership.

Analysis

Data Exploration

The `OnlineNewsPopularity.csv` file has 39,644 rows and 58 columns of predictive data, with one column of target data – `shares`. The `OnlineNewsPopularity.names` file describes the

predictive features thus:

```
n_tokens_title:                 Number of words in the title
n_tokens_content:               Number of words in the content
n_unique_tokens:                Rate of unique words in the content
n_non_stop_words:               Rate of non-stop words in the content
n_non_stop_unique_tokens:       Rate of unique non-stop words in the content
num_hrefs:                      Number of links
num_self_hrefs:                 Number of links to other articles published by Mashable
num_imgs:                       Number of images
num_videos:                     Number of videos
average_token_length:           Average length of the words in the content
num_keywords:                   Number of keywords in the metadata
data_channel_is_lifestyle:      Is data channel 'Lifestyle'?
data_channel_is_entertainment: Is data channel 'Entertainment'?
data_channel_is_bus:            Is data channel 'Business'?
data_channel_is_socmed:         Is data channel 'Social Media'?
data_channel_is_tech:           Is data channel 'Tech'?
data_channel_is_world:          Is data channel 'World'?
kw_min_min:                     Worst keyword (min. shares)
kw_max_min:                     Worst keyword (max. shares)
kw_avg_min:                     Worst keyword (avg. shares)
kw_min_max:                     Best keyword (min. shares)
kw_max_max:                     Best keyword (max. shares)
kw_avg_max:                     Best keyword (avg. shares)
kw_min_avg:                     Avg. keyword (min. shares)
kw_max_avg:                     Avg. keyword (max. shares)
kw_avg_avg:                     Avg. keyword (avg. shares)
self_reference_min_shares:      Min. shares of referenced articles in Mashable
self_reference_max_shares:      Max. shares of referenced articles in Mashable
self_reference_avg_shares:      Avg. shares of referenced articles in Mashable
weekday_is_monday:              Was the article published on a Monday?
weekday_is_tuesday:             Was the article published on a Tuesday?
weekday_is_wednesday:           Was the article published on a Wednesday?
weekday_is_thursday:            Was the article published on a Thursday?
weekday_is_friday:              Was the article published on a Friday?
weekday_is_saturday:            Was the article published on a Saturday?
weekday_is_sunday:              Was the article published on a Sunday?
is_weekend:                     Was the article published on the weekend?
LDA_00:                         Closeness to LDA topic 0
LDA_01:                         Closeness to LDA topic 1
LDA_02:                         Closeness to LDA topic 2
LDA_03:                         Closeness to LDA topic 3
LDA_04:                         Closeness to LDA topic 4
global_subjectivity:            Text subjectivity
global_sentiment_polarity:      Text sentiment polarity
global_rate_positive_words:     Rate of positive words in the content
global_rate_negative_words:     Rate of negative words in the content
rate_positive_words:            Rate of positive words among non-neutral tokens
rate_negative_words:            Rate of negative words among non-neutral tokens
avg_positive_polarity:          Avg. polarity of positive words
min_positive_polarity:          Min. polarity of positive words
max_positive_polarity:          Max. polarity of positive words
avg_negative_polarity:          Avg. polarity of negative words
min_negative_polarity:          Min. polarity of negative words
```

```
max_negative_polarity:         Max. polarity of negative words
title_subjectivity:            Title subjectivity
title_sentiment_polarity:      Title polarity
abs_title_subjectivity:        Absolute subjectivity level
abs_title_sentiment_polarity:  Absolute polarity level
```

Note: LDA is Latent Dirichlet Allocation, a natural language processing technique for discovering topics in text.[3]

This project is not about attempting to predict the number of shares; rather, it attempts to predict whether a particular article will be popular or not. So instead of using the `shares` column directly, I'll create a new column named `high_shares` based on the `shares` column. The value of the `high_shares` column will be 1 if the number of shares is greater than the median shares, or 0 otherwise. This will be the new target column.

One column (`timedelta`) is non-predictive, but answers an important question about the data. According to the column description in `OnlineNewsPopularity.names`, the `timedelta` column is "Days between the article publication and the dataset acquisition." Some articles could possibly have more shares simply because of the passage of time, so it is possible that not all articles in the dataset are on equal footing. To test whether I should remove some rows, I'll need to check `high_shares` vs. `timedelta`.

Checking for null or missing values reveals that there are none in the dataset.

Unfortunately, while sampling the Mashable articles' content for consistency with the dataset, I discovered several flaws in the data. The `num_imgs` and `num_videos` features were not accurate on any of the 10 or so articles that I sampled.

In addition, I found inaccuracies in the `self_reference_min_shares`, `self_reference_max_shares`, and `self_reference_avg_shares`. For example, for article GaymerX: World's First LGBT Gamer Convention Comes to San Francisco the value for each of these three features is 843,300. Thus one would expect to find in this article a link to a very popular Mashable article which has 843,300 shares. There is only one such article in the dataset with that many shares, and there is no link to it in the GaymerX article. Similarly, the article Sprint's New Plans Guarantee Unlimited Data for Life garnered 210,300 shares. It was the only article with that share count in the dataset. The article titled Spike Lee Wants Kickstarter's Help for His 'Newest, Hottest Joint' has 210,300 for the value in the three related features listed above. But there is no reference in the Spike Lee article to the Sprint Unlimited Data article.

The extent of the erroneous references discovered above is unknown, so I will not be removing any data.

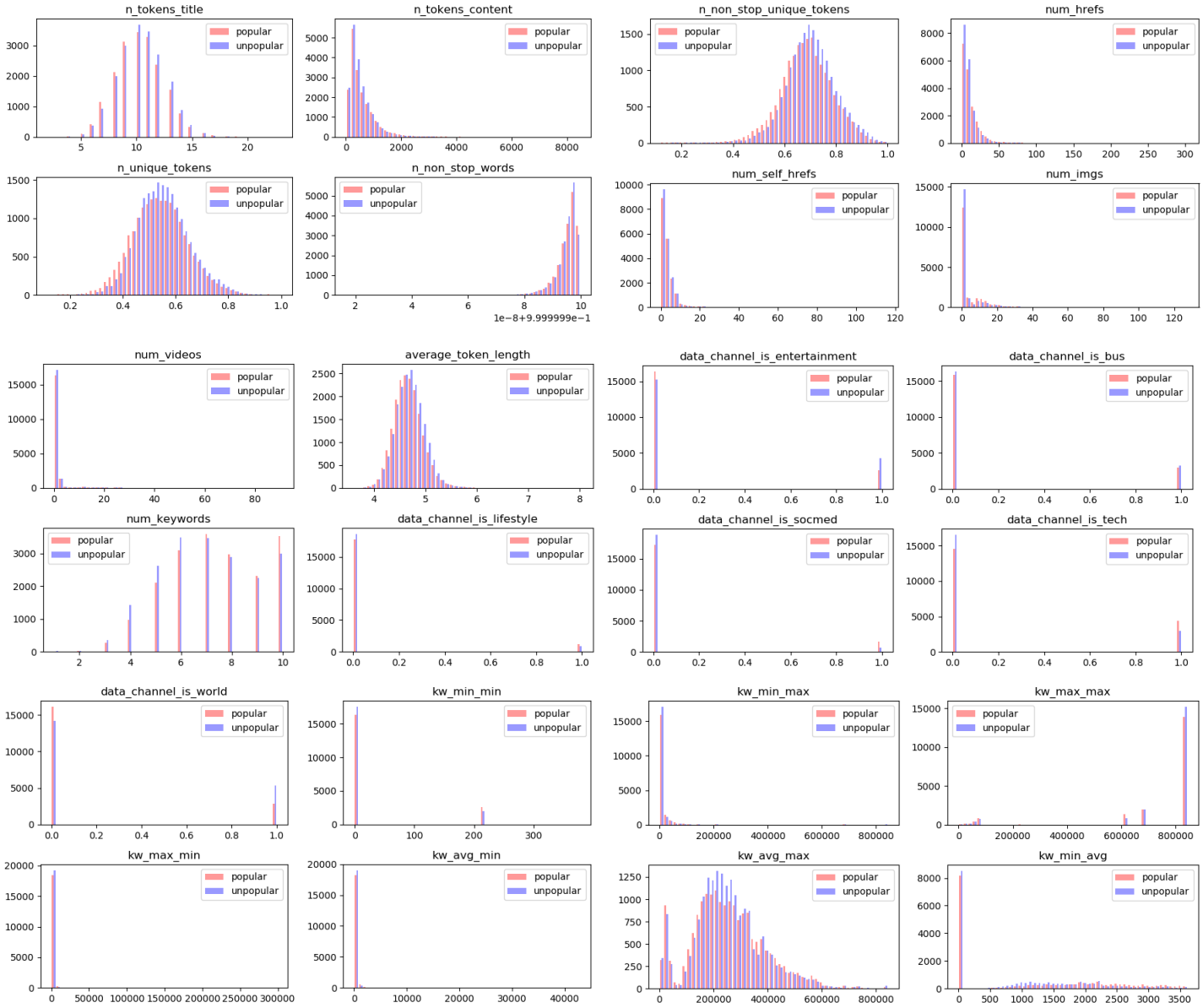
But as a result of my search for anomalous data, I will need to remove 1,181 rows which have a value of 0 in the feature `n_tokens_content`. (These rows also have 0 in a number of other features.) These data are clearly flawed. If an article has no content, then there is nothing to analyze. In addition, when I sampled the actual article content for these rows, I found that there was indeed content for the articles in question (i.e., they were not articles that merely showed images or videos with no content). I also will need to remove one row which has values greater than 1.0 for the features which are rates (e.g., `n_unique_tokens`, `n_non_stop_words`) and should therefore be under 1.0.

3 https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

I did not sample the Mashable articles' content for consistency across all of the predictive features. But having found the above inconsistencies, I can only conclude that unfortunately there are serious flaws with the dataset. There are likely other problems in addition to those listed above. Nevertheless, I will proceed with the project using the given dataset (minus the rows deleted above). This amounts to merely an academic exercise, unfortunately, as any conclusions will be just as flawed as the dataset.

Exploratory Visualization

To get a feel for how the data is distributed, the matrix below is a dual histogram analysis for each of the 58 predictive features, grouped by the `high_shares` target value. If the red and blue lines are skewed in different directions, then that feature is likely one that can help us determine popularity.




kw_max_avg
kw_avg_avg
self_reference_avg_sharess
weekday_is_monday
self_reference_min_shares
self_reference_max_shares
weekday_is_tuesday
weekday_is_wednesday
weekday_is_thursday
weekday_is_friday
is_weekend
LDA_00
weekday_is_saturday
weekday_is_sunday
LDA_01
LDA_02
LDA_03
LDA_04
global_rate_positive_words
global_rate_negative_words
global_subjectivity
global_sentiment_polarity
rate_positive_words
rate_negative_words
avg_positive_polarity
min_positive_polarity
min_negative_polarity
max_negative_polarity
max_positive_polarity
avg_negative_polarity
title_subjectivity
title_sentiment_polarity
abs_title_subjectivity
abs_title_sentiment_polarity
popular
unpopular

The above matrix does not reveal any significant skewing, so the machine learning classifiers will not have an easy job.

Algorithms and Techniques

This problem is a single class supervised classification problem, so much like Fernandes, et al., I will be using a variety of classification algorithms to determine which one does the best job of classifying the validation data. I will try:

K-Nearest Neighbors (KNN)

KNN is a simple algorithm that predicts by matching the current sample to its nearest labeled sample using a similarity measure such as a distance function.

Support Vector Machine (SVM)

SVM looks at the training data and calculates two things – a classification error and a margin error, and then it uses gradient descent iteratively to minimize the sum of those errors. It draws a line (technically a hyperplane) through the data points and tries to minimize not just the distances of the miss-classified points to the line, it also draws an upper and lower margin around that line and considers the points within the margin to be miss-classified as well. It thus rewards data points further from the boundary line and punishes those points that are either miss-classified or are too close to the boundary line. And SVM doesn't restrict itself to linear separation; it also utilizes polynomial functions to find the best equation for separating classes.

Random Forest

A Random Forest classifier uses a multitude of Decision Trees to make its predictions. Each Decision Tree is similar to a series of True/False questions, where the questions derive from the features of the dataset. One Decision Tree might be able to make a decent prediction, but the Random Forest algorithm trains many Decision Trees on the training data as a way of increasing the chances of arriving at more accurate predictions.

AdaBoost

Similar to Random Forest, AdaBoost (short for Adaptive Boosting) uses multiple classifiers to create a more accurate prediction. The classifiers here are “weak learners” because they only need to be slightly better than random guessing. This ensemble learning technique iteratively updates the classifier weights and the feature weights to emphasize the miss-classified data points, and de-emphasize the correctly classified data points.

XGBoost

XGBoost (short for Extreme Gradient Boosting) is similar to AdaBoost in employing an ensemble learning technique where new models are created that predict the errors of prior models and are then added together to make the final prediction. It is called gradient boosting because it uses gradient descent to minimize the loss when adding new models.[4]

Gaussian Naive Bayes

Gaussian Naive Bayes assumes that each feature of the dataset is independent of the other features. It further assumes that the features follow a Gaussian, or normal, distribution. In simplified terms, the Naive Bayes classifier uses the training samples to calculate probabilities for each of the features versus the target variable. It then multiplies all of those probabilities

4 Jason Brownlee, Ph.D., “A Gentle Introduction to XGBoost for Applied Machine Learning,” August 16, 2017, https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.

together for each class of target outcome and whichever outcome has the highest score is the prediction.

For this particular project, I believe AdaBoost and XGBoost are excellent choices because they both do very well in all types of binary classification situations. Random Forest is also a good choice for similar reasons. All three use multiple learners and then find a solution by aggregating the wisdom of the ensemble. KNN is a good choice because of its simplicity. It provides a good baseline that might actually be competitive with the ensemble learners. SVM could do well since it can handle non-linearly separable classes and will not get stuck in a local minimum. Naive Bayes has less of a chance to shine in this problem because not all of the features are uncorrelated. All of these algorithms except for XGBoost were used by Fernandes, et al. in their study, so I believe it is important to include all of them here for comparison's sake.

For each classifier, I'll use Scikit-Learn's GridSearch to tune the hyperparameters, then test the best configuration of each algorithm against the validation set. I'll also test three different scalers from Scikit-Learn: the Standard scaler, the Min-Max scaler and the Robust scaler to see which one results in the best predictions.

In addition to the classifiers above, I'll also test a Neural Network approach, varying the configurations to find the best hyperparameters.

Lastly, when I've found the best classifier with the optimal configuration, I'll extract the feature importances to see which features are judged to be the most important when targeting a high number of shares.

Benchmark

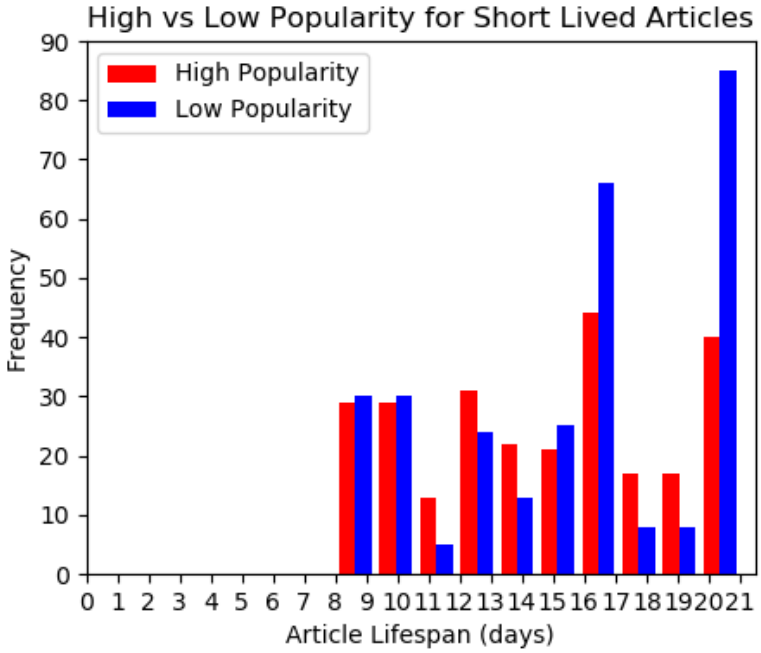
Fernandes, Vinagre and Cortez in their study achieved an accuracy of 67%, an F1 score of 0.69 and an area under the Receiver Operating Characteristic curve of 73%[5]. This was achieved using a rolling window approach (10,000 training samples vs 1,000 test samples). I will be taking a more traditional approach of dividing up the dataset into training, validation and testing sets. Nevertheless, I aim to achieve similar results.

Methodology

Data Preprocessing

Below is a histogram analysis of popularity for articles that have only had a lifetime of 21 days or less. It seems that the short lived articles are not significantly skewed toward low shares, so I will not remove any rows.

5 K. Fernandes, P. Vinagre and P. Cortez. A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News. Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence, September, Coimbra, Portugal, 8.



As mentioned above in the Data Exploration section, I removed 1,181 rows which have a value of 0 in the feature `n_tokens_content`. I also removed one row which had values greater than 1.0 for the rate-based features and therefore must be under 1.0. This brought my total number of rows down to 38,462. Since Fernandes, Vinagre and Cortez used 39,000 articles in their dataset[6], I must assume that they did not remove these errant data.

The median number of shares for the entire dataset is 1,400. Fernandes, et al. considered a popular article as one which had shares greater than the median.[7] My dataset of 38,462 articles breaks down as follows:

shares > median	18,911
shares < median	17,999
shares == median	1,552

To split the data as equitably as possible, I'll consider popular articles to be greater than the median, and unpopular articles less than or equal to the median. Fernandes, et al. did not specify how they classified articles which had shares equal to the median.

Implementation

I began the implementation by using the Python library Scikit-Learn to re-scale the data. I used the Standard Scaler (`sklearn.preprocessing.StandardScaler`) as my default, although I also tested `sklearn.preprocessing.MinMaxScaler` and `sklearn.preprocessing.RobustScaler` while refining my results.

I then split the data into training, validation and testing sets by first running `sklearn.model_selection.train_test_split`:

6 Ibid., 3.
7 Ibid., 7.

```
X_train, X_test, y_train, y_test =
     train_test_split(X, y, test_size=0.25, random_state=42)
```

to create a test set equal to 25% of the data, while the training set (`X_train, y_train`) was equal to 75% of the data. I then split the training data again:

```
X_train, X_val, y_train, y_val =
     train_test_split(X_train, y_train, test_size=0.3333, random_state=42)
```

to create a validation set which was one-third the size of the training set, meaning that the training set was not (0.6667 * 0.75) = 0.5 and the validation set was (0.3333 * 0.75) = 0.25. I now had a split of 50% training, 25% test and 25% validation data.

I ran an initial baseline test against each of the classifiers using their default hyperparameters. They were all trained on the training set, and tested against the validation set. The six classifiers came from the following sources:

- `sklearn.neighbors.KNeighborsClassifier`
- `sklearn.svm.SVC`
- `sklearn.ensemble.RandomForestClassifier`
- `sklearn.ensemble.AdaBoostClassifier`
- `xgboost.XGBClassifier`
- `sklearn.naive_bayes.GaussianNB`

To train each of the classifiers I ran the `fit` method using the training data for the arguments. To generate predictions I ran `predict`, using the validation data as arguments. To calculate the five scores, I used:

- `sklearn.metrics.accuracy_score`
- `sklearn.metrics.precision_score`
- `sklearn.metrics.recall_score`
- `sklearn.metrics.f1_score`
- `sklearn.metrics.roc_auc_score`

Below are the results.

KNN

Default Model Score on Validation Set	
Accuracy	0.6111
Precision	0.6048
Recall	0.5664
F1	0.5850
AUC of ROC	0.6097

SVM

Default Model Score on Validation Set	
Accuracy	0.6570
Precision	0.6500
Recall	0.6307
F1	0.6402
AUC of ROC	0.6562

RandomForest

Default Model Score on Validation Set	
Accuracy	0.6251
Precision	0.6334
Recall	0.5344
F1	0.5797
AUC of ROC	0.6222

AdaBoost

Default Model Score on Validation Set	
Accuracy	0.6612
Precision	0.6496
Recall	0.6507
F1	0.6501
AUC of ROC	0.6608

XGBoost

Default Model Score on Validation Set	
Accuracy	0.6707
Precision	0.6601
Recall	0.6584
F1	0.6593
AUC of ROC	**0.6703**

Naive Bayes

Default Model Score on Validation Set	
Accuracy	0.6070
Precision	0.6394
Recall	0.4304
F1	0.5145
AUC of ROC	0.6014

XGBoost produced the best results, with an AUC of ROC score of 0.6703.

I also ran an initial baseline test with a neural net using Keras. Keras does not allow the user to specify AUC of ROC as the metric to be evaluated by the model during training and testing. The default choice is the accuracy score, which correlates very well with AUC of ROC; so for my purposes, this was only a minor issue.

My initial neural net was comprised of 2 fully connected layers with two Dropout layers, followed by the output layer:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                1180
_________________________________________________________________
dropout_1 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
_________________________________________________________________
dropout_2 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 22
=================================================================
Total params: 1,412
Trainable params: 1,412
Non-trainable params: 0
```

I set up the model training to test against the validation set after each of 300 epochs, with a batch size of 32, a `sparse_categorical_crossentropy` loss function, and `adam` optimizer. I had it track the best AUC of ROC on the validation set, and when it found a new best value, it also recorded the Accuracy, Precision, Recall and F1 scores for that particular epoch. So the figures in the table below all come from the same training epoch (epoch 144). They are not the top scores achieved from different epochs for each of the different scores.

Neural Net

Default Model Score on Validation Set	
Accuracy	0.6602
Precision	0.6375
Recall	0.6900
F1	0.6627
AUC of ROC	0.6612

Refinement

The following table pairs summarize the process of refining the hyperparameters of each classifier by implementing GridSearch. The Optimized Model Score in the bottom table of the pair records the score against the validation set using the best hyperparameter values discovered via GridSearch and presented in the top table of the pair. My objective is to maximize the AUC of the ROC, so when evaluating the best data scaler, the top choice is the scaler which produces the highest AUC of ROC score.

KNN

Hyperparameter	Values Tested	Best
weights	'uniform', 'distance'	'distance'
n_neighbors	1, 5, 10, 15, 20, 30, 35, 36, 37, 38, 39, 40, 41, 43, 45, 50, 60	37
algorithm	'auto', 'ball_tree', 'kd_tree', 'brute'	'auto'

Optimized Model Score Validation Set			
	Standard Scaler	MinMax Scaler	Robust Scaler
Accuracy	0.6361	0.6271	0.6236
Precision	0.6396	0.6202	0.6167
Recall	0.5677	0.5916	0.5866
F1	0.6015	0.6056	0.6013
AUC of ROC	0.6339	0.6260	0.6225

SVM

Hyperparameter	Values Tested	Best
gamma	1, 2, 3, 4	1
C	1, 2, 3, 4	2
kernel	'linear', 'poly', 'rbf', 'sigmoid'	'linear'

Optimized Model Score Validation Set			
	Standard Scaler	MinMax Scaler	Robust Scaler
Accuracy	0.6384	0.6263	0.5764
Precision	0.6256	0.6140	0.5482
Recall	0.6290	0.6131	0.7074
F1	0.6273	0.6135	0.6177
AUC of ROC	0.6381	0.6259	0.5805

RandomForest

Hyperparameter	Values Tested	Best
max_depth	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18, 20, 25	17
n_estimators	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 650, 700	600
criterion	'gini', 'entropy'	'entropy'
max_features	'auto', 'log2', 3, 5, 6, 7, 8, 10, 12, 15, 20	'auto'
min_samples_split	3, 5, 10, 15, 17, 19, 20, 21, 23, 25, 30	20

Optimized Model Score Validation Set			
	Standard Scaler	MinMax Scaler	Robust Scaler
Accuracy	0.6713	0.6723	0.6724
Precision	0.6580	0.6584	0.6591
Recall	0.6679	0.6707	0.6687
F1	0.6629	0.6645	0.6639
AUC of ROC	0.6712	0.6722	0.6723

AdaBoost

Hyperparameter	Values Tested	Best
base_estimator	DecisionTree Classifier(default config), RandomForest Classifier(above config)	RandomForest Classifier(above config)
n_estimators	25, 50, 100, 300	25
learning_rate	0.01, 0.1, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4	1.3
algorithm	'SAMME.R', 'SAMME'	'SAMME.R'

Optimized Model Score Validation Set			
	Standard Scaler	MinMax Scaler	Robust Scaler
Accuracy	0.6759	0.6757	0.6756
Precision	0.6611	0.6612	0.6611
Recall	0.6773	0.6763	0.6761
F1	0.6691	0.6687	0.6685
AUC of ROC	0.6760	0.6757	0.6756

XGBoost

Hyperparameter	Values Tested	Best
max_depth	2, 3, 4, 5, 6	5
learning_rate	0.03, 0.04, 0.05, 0.07, 0.75, 0.1	0.05
n_estimators	50, 100, 150, 200, 300, 400	200
gamma	0.0, 0.1, 0.25, 0.5, 1.0, 1.5, 2.0, 5.0	0.25
subsample	0.6, 0.8, 0.9, 1.0	0.8
colsample_bytree	0.6, 0.8, 0.9, 0.95, 1.0	0.95

Optimized Model Score Validation Set			
	Standard Scaler	MinMax Scaler	Robust Scaler
Accuracy	0.6786	0.6768	0.6782
Precision	0.6667	0.6652	0.6658
Recall	0.6713	0.6681	0.6724
F1	0.6690	0.6667	0.6691
AUC of ROC	**0.6784**	0.6765	0.6780

Naive Bayes

Note: The Naive Bayes Classifier does not have any hyperparameters to tune.

Note: For the Neural Net, I tested out the configurations manually.

Neural Net

Hyperparameter	Values Tested	Best
batch_size	16, 32, 48, 64	32
loss	'sparse_categorical_crossentropy', 'mse', 'binary_crossentropy'	'binary_crossentropy'
optimizer	'adam', 'adamax', 'nadam', 'adadelta', 'adagrad'	'adam'
Activation	'relu', 'tanh', 'elu'	'relu'
Dropout rate	0.3, 0.4, 0.5, 0.55, 0.6, 0.65, 0.7	0.6
No. of Layers	1, 2, 3	3
Layer 1 Units	20, 40, 50, 60, 70	40
Layer 2 Units	10, 15, 20, 25, 30, 40	20
Layer 3 Units	5, 10, 15, 20	10

Optimized Model Score Validation Set			
	Standard Scaler	**MinMax Scaler**	**Robust Scaler**
Accuracy	0.6647	0.6577	0.6427
Precision	0.6543	0.6459	0.6021
Recall	0.6509	0.6475	0.7713
F1	0.6526	0.6467	0.6763
AUC of ROC	0.6643	0.6574	0.6468

Results

Model Evaluation and Validation

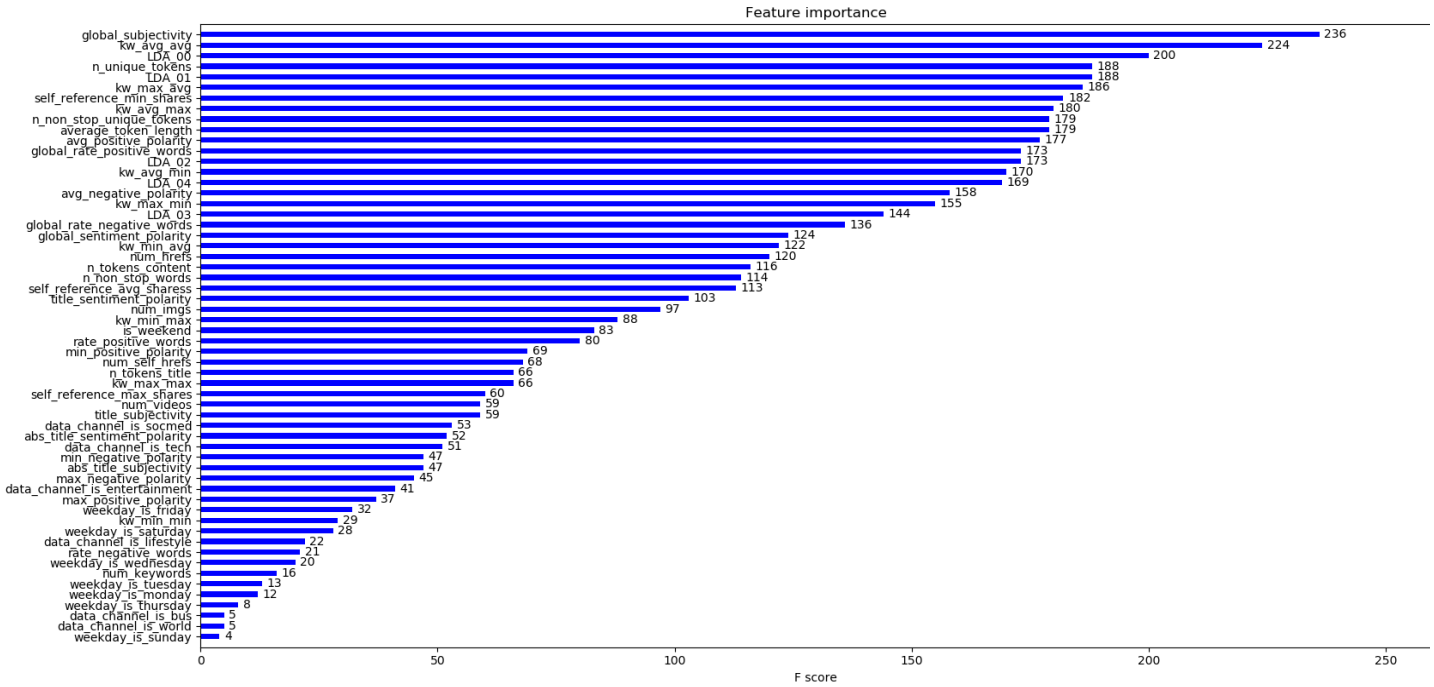
The table below summarizes the performance of the above classifiers on the validation data.

Classifier	AUC of ROC
KNN	0.6339
SVM	0.6381
RandomForest	0.6723
AdaBoost	0.6760
XGBoost	**0.6784**
Naive Bayes	0.6014
Neural Net	0.6643

The best performing classifier was XGBoost using Scikit-Learn's Standard Scaler to scale the data. It achieved an AUC of the ROC of 0.6784. The hyperparameter configuration was:

XGBoost Hyperparameter	Best
max_depth	5
learning_rate	0.05
n_estimators	200
gamma	0.25
subsample	0.8
colsample_bytree	0.95

With this configuration I was able to determine the most important features, using XGBoost's `plot_importance` function. The image below summarizes the results:



The highest ranked features are:

Feature	Score
Article text subjectivity	236
Avg. keyword (avg. shares)	224
Closeness to top 1 LDA topic	200
Rate of unique words	188
Closeness to top 2 LDA topic	188
Avg. keyword (max. shares)	186
Min. shares of Mashable links	182
Best keyword (avg. shares)	180
Rate of unique non-stop words	179
Average word length	179

Each classifier's performance against the test data vs. the validation data is summarized in the table below. All of the models show only a slight difference between the validation set and the test set, which is data completely unknown to the models.

Validation vs Test										
	Accuracy		Precision		Recall		F1		AUC	
Model	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
KNN	0.6361	0.6334	0.6396	0.6420	0.5677	0.5740	0.6015	0.6061	0.6339	0.6324
SVM	0.6384	0.6418	0.6256	0.6375	0.6290	0.6284	0.6273	0.6329	0.6381	0.6416
RandomForest	0.6713	0.6624	0.6580	0.6585	0.6679	0.6504	0.6629	0.6544	0.6712	0.6622
AdaBoost	0.6759	**0.6694**	0.6611	**0.6651**	**0.6773**	**0.6590**	**0.6691**	**0.6621**	0.6760	**0.6692**
XGBoost	**0.6786**	0.6657	**0.6667**	0.6633	0.6713	0.6491	0.6690	0.6561	**0.6784**	0.6654
Naive Bayes	0.6070	0.6104	0.6394	0.6618	0.4304	0.4237	0.5145	0.5166	0.6014	0.6073
Neural Net	0.6647	0.6611	0.6543	0.6589	0.6509	0.6434	0.6526	0.6510	0.6643	0.6608

All of the models seem quite robust. XGBoost and AdaBoost scored best against the validation data alone, and if forced to choose between the two, the AdaBoost model would appear to be slightly more robust. For AUC of ROC, AdaBoost achieved 0.6760 vs the validation set and 0.6692 vs the test set, the difference being a scant 0.0068. XGBoost achieved 0.6784 vs the validation set and 0.6654 vs the test set, for a difference of 0.0130 – a very robust model, but not quite as good as AdaBoost. Against the accuracy score, they did similarly. AdaBoost (0.6759 validation, 0.6694 test → 0.0065 difference) did slightly better than XGBoost (0.6786 validation, 0.6657 test → 0.0129 difference).

Justification

The table below compares the top feature importances I found vs. the rankings for those features from Fernandes, et. al.[8] Considering that there were 58 total features, I was pleasantly surprised to find that the top ten features from this project were all included in Fernandes's top fifteen, even though the Fernandes team used a rolling window approach while I used a more traditional approach.

Feature	Rank	Fernandes Rank
Article text subjectivity	1	13
Avg. Keyword (avg. shares)	2	1
Closeness to top 1 LDA topic	3	11
Rate of unique words	4	14
Closeness to top 2 LDA topic	5	8
Avg. keyword (max. shares)	6	2
Min. shares of Mashable links	7	5
Best keyword (avg. shares)	8	6
Rate of unique non-stop words	9	12
Average word length	10	15

The table below summarizes the predictive values of each classifier against the validation set, the test set, and against the reported values of Fernandes, et al. The red highlighted numbers are the best scores achieved against the validation set, the purple numbers are the best scores achieved against the test set, and the blue numbers are the best figures that Fernandes, et al. achieved.

8 Ibid., 9

Validation / Test / Fernandes, et al.					
Model	**Accuracy**	**Precision**	**Recall**	**F1**	**AUC**
KNN	0.64/0.63/0.62	0.64/0.64/0.66	0.57/0.57/0.55	0.60/0.61/0.60	0.63/0.63/0.67
SVM	0.64/0.64/0.66	0.63/0.64/0.67	0.63/0.63/0.68	0.63/0.63/0.68	0.64/0.64/0.71
RandomForest	0.67/0.66/**0.67**	0.66/0.66/0.67	0.67/0.65/**0.71**	0.66/0.65/**0.69**	0.67/0.66/**0.73**
AdaBoost	0.68/**0.67**/0.66	0.66/**0.67**/0.68	**0.68**/**0.66**/0.67	**0.67**/**0.66**/0.67	0.68/**0.67**/0.72
XGBoost	**0.68**/0.67/ NA	**0.67**/0.66/ NA	0.67/0.65/ NA	0.67/0.66/ NA	**0.68**/0.67/ NA
Naive Bayes	0.61/0.61/0.62	0.64/0.66/**0.68**	0.43/0.42/0.49	0.51/0.52/0.57	0.60/0.61/0.65
Neural Net	0.67/0.66/ NA	0.66/0.66/ NA	0.66/0.64/ NA	0.66/0.65/ NA	0.67/0.66/ NA

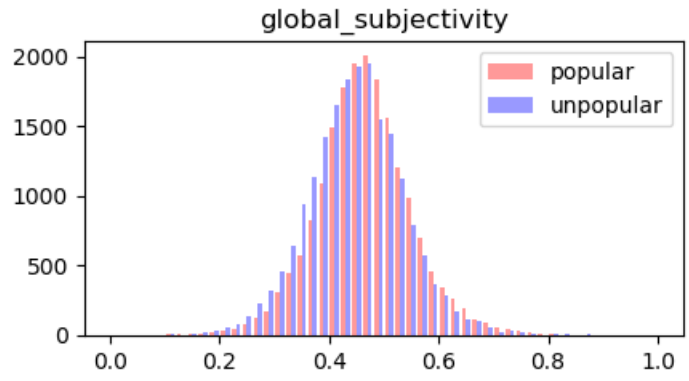
AdaBoost and XGBoost are the clear winners in this project. XGBoost had a slight advantage against the validation set, but AdaBoost did the best in all categories against the test set, which is where our focus lies.

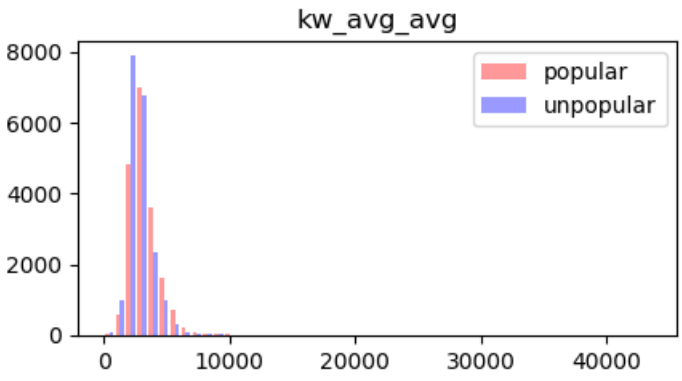
When compared to the numbers achieved by Fernandes, et al. and their rolling windows approach, the numbers achieved in this project are not as good for AUC of the ROC. The accuracy scores are slightly better than Fernandes, and the Precision and F1 scores are quite comparable. Overall, since the numbers are not strictly comparable, it is gratifying to see the numbers achieved in this project approach those of Fernandes, et al.

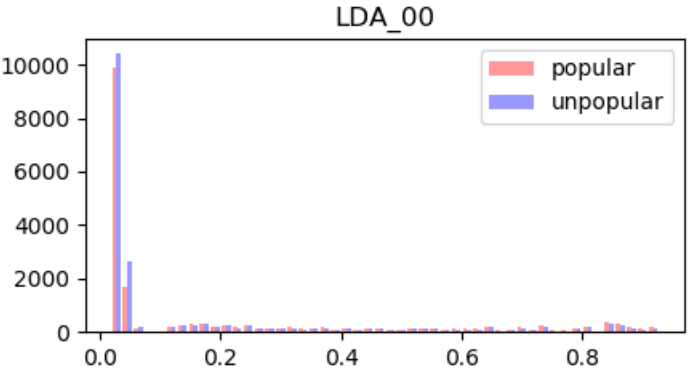
Conclusion

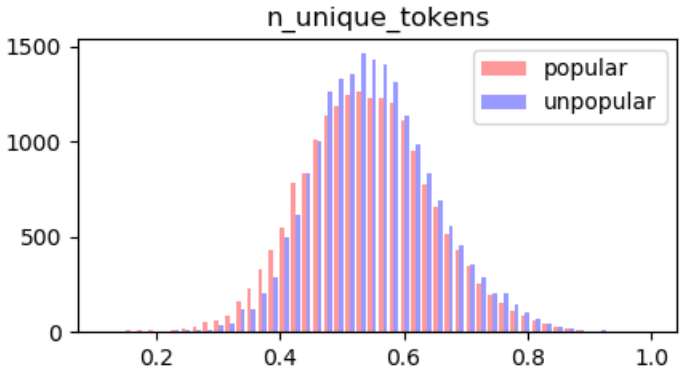
Free-Form Visualization

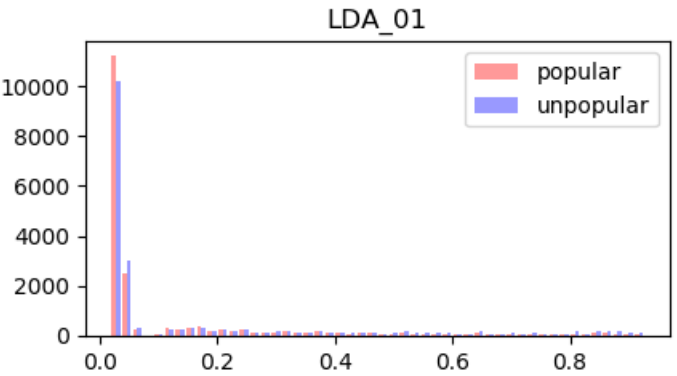
Below are a set of histograms, repeated from above, but this time showing only the top ten features. Each of them shows a notable, though slight, skewing of the popular and unpopular distributions. When combined, these skewings enable the classifier to make decisions about how to classify a particular sample. If there were no noticeable skewings of the distributions, the classifier's accuracy would be greatly diminished.

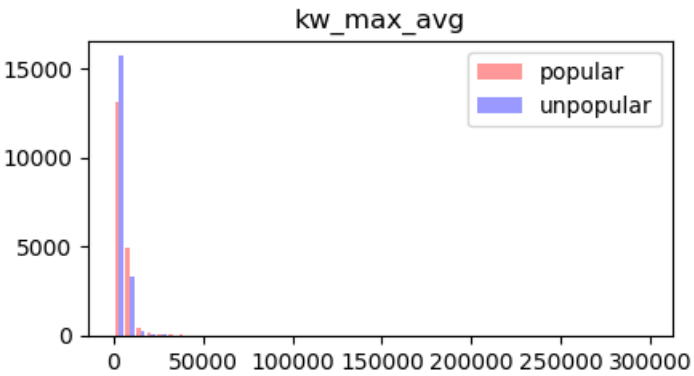

global_subjectivity
popular
unpopular
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0

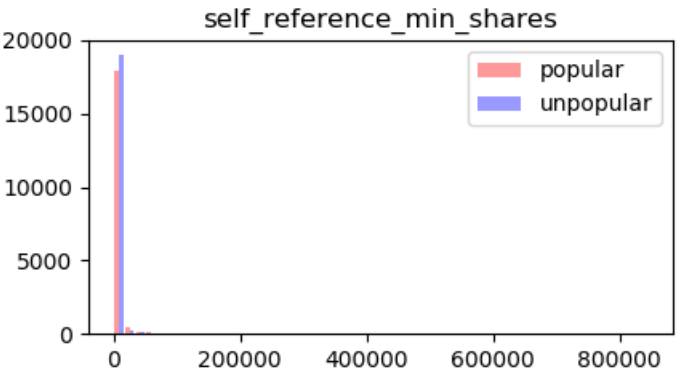

kw_avg_avg
popular
unpopular
8000
6000
4000
2000
0
0
10000
20000
30000
40000

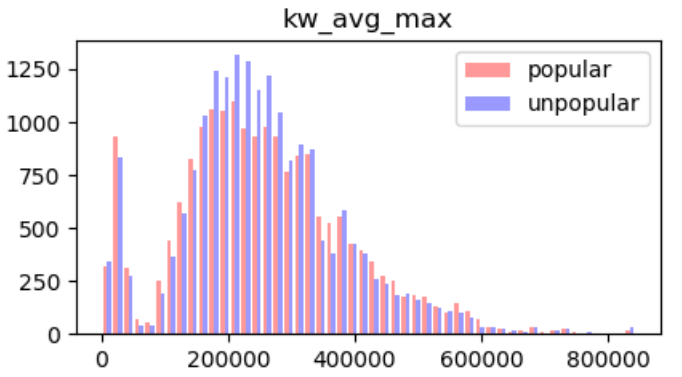

LDA_00
popular
unpopular
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8

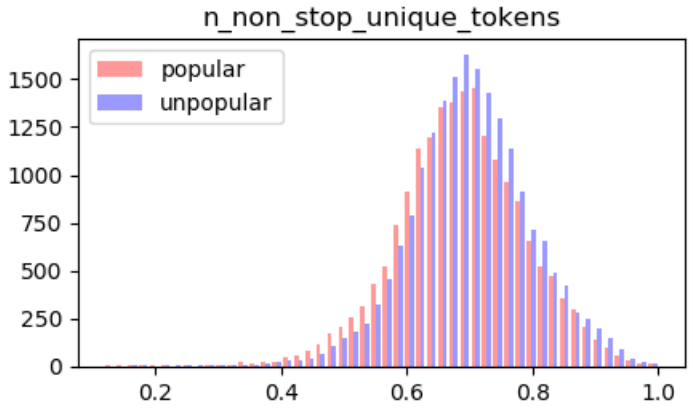

n_unique_tokens
popular
unpopular
1500
1000
500
0
0.2
0.4
0.6
0.8
1.0

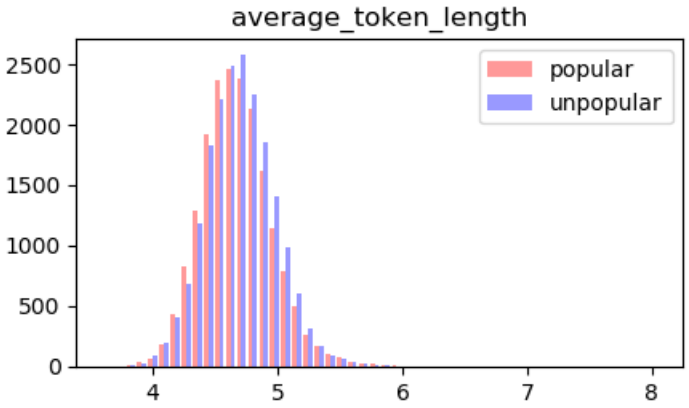

LDA_01
popular
unpopular
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8


kw_max_avg
popular
unpopular
15000
10000
5000
0
0
50000
100000
150000
200000
250000
300000


self_reference_min_shares
popular
unpopular
20000
15000
10000
5000
0
0
200000
400000
600000
800000

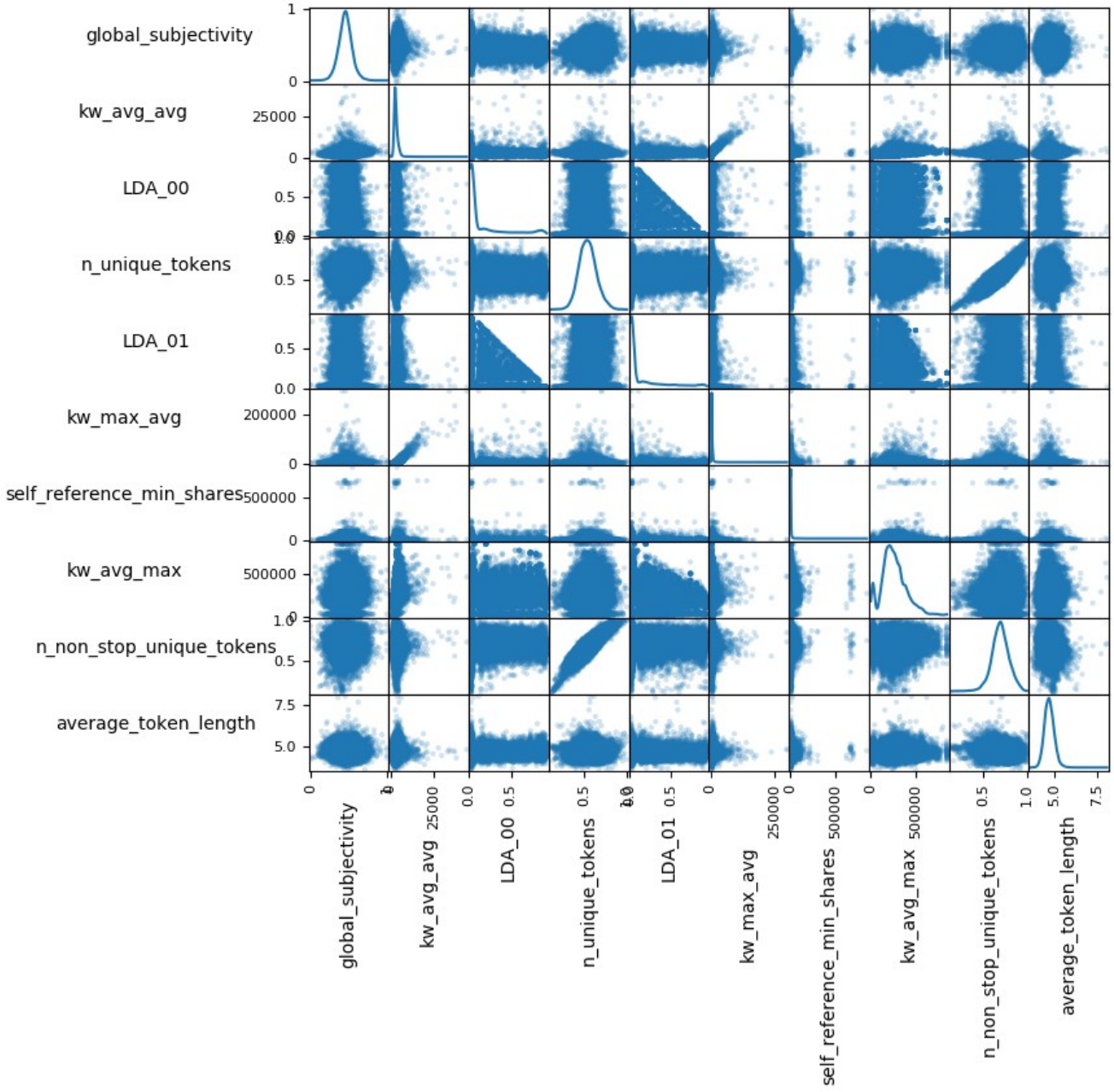

kw_avg_max
popular
unpopular
1250
1000
750
500
250
0
0
200000
400000
600000
800000





Below is a scatter plot matrix comparing feature to feature for these ten features. Most of the top ten do not show any significant correlation with other top ten features. This is to be expected. If the features correlated nicely with other features, those features would be largely redundant. So a lack of correlation is what we would expect to see in a set of features that XGBoost regards as the ten most important.

There is positive correlation between `n_unique_tokens` and `n_non_stop_unique_tokens` – i.e., “Rate of unique words” and “Rate of unique non-stop words”. There is also a positive correlation between `kw_avg_avg` and `kw_max_avg` – i.e., “Avg. Keyword (avg. shares)” and “Avg. keyword (max. shares).” But this is still a relatively small amount of correlation among the top features.



Reflection

The entire process can be summarized in the following steps:

1. Download the `OnlineNewsPopularity.csv` file
2. Check for null values and bad data
3. Formally define popular/unpopular for target feature
4. Create target feature `high_shares`
5. Explore histogram analysis of features
6. Check validity of short lived articles

7. Re-scale the data
8. Split the data into train, test, and validation sets
9. Run initial baseline test against each classifier and neural net using default hyperparameters
10. Tune classifier hyperparameters to find best AUC of ROC scores
11. Calculate top ten feature importances using best classifier
12. Test all best classifiers and neural net against the test dataset
13. Create histograms of top ten features
14. Create scatter matrix of top ten features

The best classifiers I found were XGBoost and AdaBoost, while Fernandes, et al. found Random Forest to be the best. And even though I chose the traditional train/validation/test split of the dataset, while Fernandes, et al. chose the rolling windows approach, our results are fairly similar. The various accuracy scores differ only by a relatively small degree, and the feature importances show a large overlap. From this narrow perspective, I see no reason to disagree with the overall conclusion reached by Fernandes, et al. Their ideas regarding editorial changes to make a news article more popular would not be contradicted by this project. But as mentioned above, the data itself is of questionable accuracy, so any conclusions should be considered preliminary at best.

The most interesting aspect of this project was the discovery of the bad data. I was so surprised to find it that I checked it multiple times to make sure I wasn't making a mistake. I opened up the `OnlineNewsPopularity.csv` file in a spreadsheet application to check the values more than once. Having seen the issues with the image and video counts and the rows of errant data, I realized there were likely other issues too. But once I had found the problems with the `self_reference_min_shares`, `self_reference_max_shares`, and `self_reference_avg_shares` columns, I stopped searching for additional errors, as there wasn't much point. This was very distressing, as it was too late to change my project. So I was forced to continue as if the data were correct, which amounts to merely an academic exercise, and not a very good one.

I am still shocked that Fernandes, et al. did not discover these errors and inconsistencies.

Improvement

The most important improvement that could be made would be a thorough examination of the dataset, comparing the results of the parsed data to the content of the articles. As mentioned above, the number of images and number of videos is very unreliable, and I was able to find other errors too. The applicability of the above results to future articles published on Mashable may be largely diminished by the quality of the data – garbage in, garbage out.

A second area where I believe there could be some improvement in the various accuracy scores is tuning XGBoost. I really only scratched the surface of the tunable parameters. XGBoost has a very impressive array of tunable parameters, and I believe that with some time and effort, I could achieve better scores against the validation and test data with XGBoost.