IE7374 FINAL PROJECT REPORT
Online New Popularity

Group 9
Monisha Prasad
Rahasya Chandan
Zheng Lu

prabhuprasad.m@northeastern.edu
chandan.r@northeastern.edu
lu.zheng1@northeastern.edu

Percentage of Effort Contributed by Student 1:_33.33%_
Percentage of Effort Contributed by Student 2:_33.33%_
Percentage of Effort Contributed by Student 3:_33.33%_
Signature of Student 1: Monisha Prasad
Signature of Student 2: Rahasya Chandan
Signature of Student 3: Zheng Lu

Submission Date: 08/08/2022

Problem Setting and Definition

With changing times all forms of media are shifting from print to digital, news is no exception. For news outlets, publishing news articles online can prove to be a step up from traditional newspapers because user interaction can be taken into consideration to assess if an article proved to be popular or not. The online news popularity dataset is a collection of certain statistics that could help us predict the popularity of online news. The dataset keeps a record of factors such as the number of words in an article, number of images, day of the week it was published on etc. The goal of our project is to use this information to extract common features among popular articles that can be put to use in future publications, and predict the extent of news popularity.

Due to the size of the dataset we plan to reduce the number of attributes by implementing dimensionality reduction. We will then be implementing regression models on the data and set threshold values for low, moderate, and high popularity based on shares. Visualizations will be used to observe patterns that good articles have in common as well as explore relationships between `shares` and other predictive attributes

Data Source

The 'Online News Popularity' dataset can be found on the UCI Machine Learning Repository - https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity

Data Description

The dataset had 39797 instances and 61 attributes. A brief description of each of those is as follows:
0. url: URL of the article
1. timedelta: Days between the article publication and the dataset acquisition
2. n_tokens_title: Number of words in the title
3. n_tokens_content: Number of words in the content
4. n_unique_tokens: Rate of unique words in the content
5. n_non_stop_words: Rate of non-stop words in the content
6. n_non_stop_unique_tokens: Rate of unique non-stop words in the content
7. num_hrefs: Number of links
8. num_self_hrefs: Number of links to other articles published by Mashable
9. num_imgs: Number of images
10. num_videos: Number of videos
11. average_token_length: Average length of the words in the content
12. num_keywords: Number of keywords in the metadata
13…18.data_channel_is_lifestyle/data_channel_is_entertainment/data_channel_is_bus/ data_channel_is_socmed/data_channel_is_tech/data_channel_is_socmed/data_channel_is_tech/ data_channel_is_world : Is data channel 'Lifestyle'/'Entertainment'/'Business'/'Social Media'/ 'Tech'/'World'?
19…21. kw_min_min/kw_max_min/kw_avg_min: Worst keyword (min/max/avg shares)
22…24. kw_min_max/kw_max_max/kw_avg_max: Best keyword (min/max/avg shares)
25…27. kw_min_avg/kw_max_avg/kw_avg_avg: Avg. keyword (min. shares)
28…30. self_reference_min_shares/self_reference_max_shares/self_reference_avg_sharess : Min/ Max/Avg. shares of referenced articles in Mashable

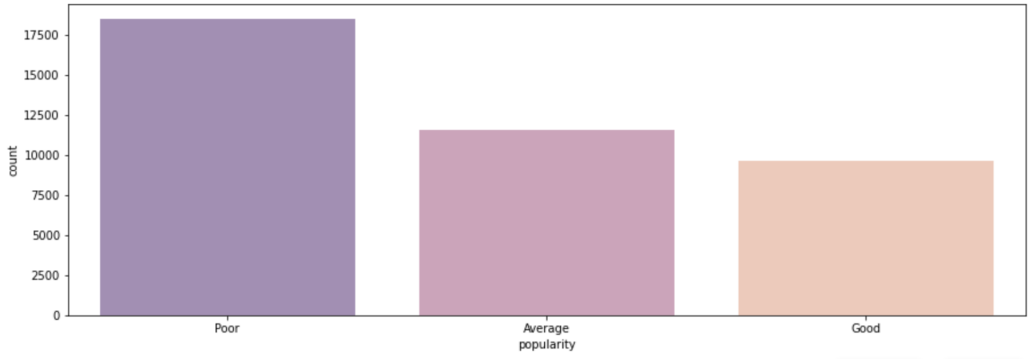
31…37. weekday_is_monday/weekday_is_tuesday/ weekday_is_wednesday/weekday_is_thursday/ weekday_is_friday/weekday_is_saturday/weekday_is_sunday: What day of week the article was published on
38. is_weekend: Was the article published on the weekend?
39…43. LDA_00/LDA_01/LDA_02/LDA_03/LDA_04: Closeness to LDA topic 0/1/2/3/4
44. global_subjectivity: Text subjectivity
45. global_sentiment_polarity: Text sentiment polarity
46. global_rate_positive_words: Rate of positive words in the content
47. global_rate_negative_words: Rate of negative words in the content
48. rate_positive_words: Rate of positive words among non-neutral tokens
49. rate_negative_words: Rate of negative words among non-neutral tokens
50…52. avg_positive_polarity/min_positive_polarity/max_positive_polarity: Avg/Min/Max polarity of positive words
53…55. avg_negative_polarity/min_negative_polarity/max_negative_polarity: Avg/Min/Max polarity of negative words
56. title_subjectivity: Title subjectivity
57. title_sentiment_polarity: Title polarity
58. abs_title_subjectivity: Absolute subjectivity level
59. abs_title_sentiment_polarity: Absolute polarity level
60. shares: Number of shares
The target variable is the ‘shares’ attribute. The ‘url’ and ‘timedelta’ variables will not be used for classification as they are non-predictive.

Data Pre-Processing

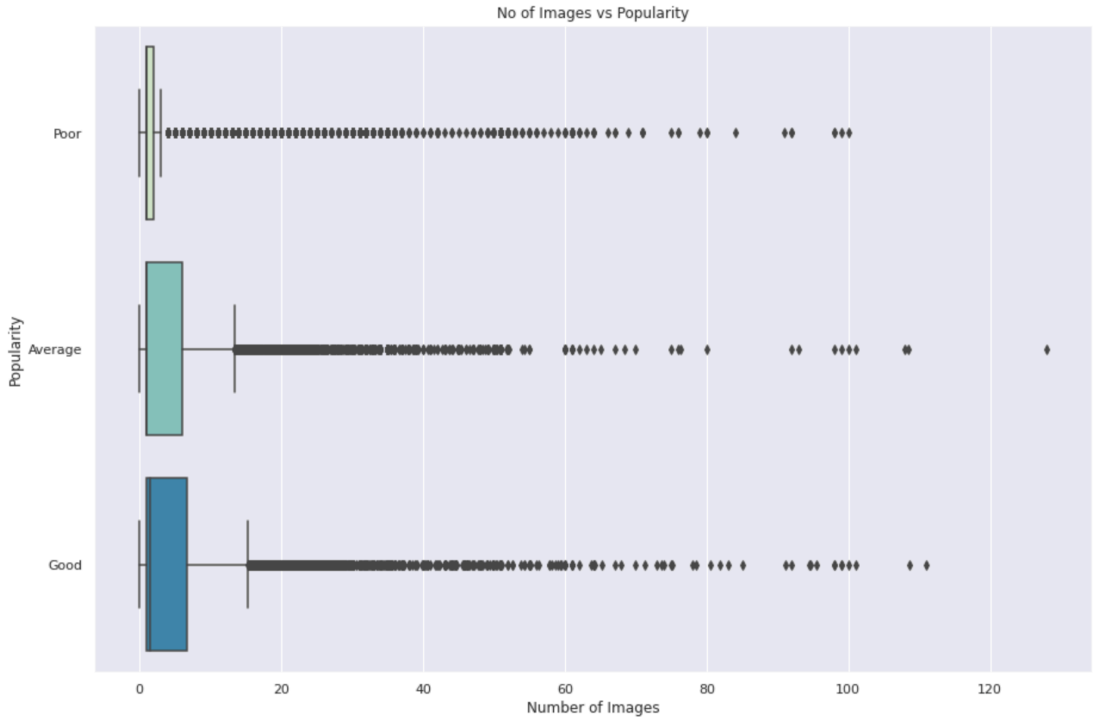
Before exploratory data analysis, we perform data pre-processing methods including:
- Checking for null values: No missing values were found
- Removing Features: remove unwanted columns like ‘url’, and ‘timedelta’ that do not provide any information about the target variable
- Adding Categories: Using .describe() the 25%, 50%, and 75% value of shares were noted and used to divided into three categories: 'Poor': below 50% ,’Average': between 50% and 75%, ’Good': above 75%
- Checking for Multicollinearity: The correlation among the variables were calculated and features such as ' n_unique_tokens', ' n_non_stop_words’ that had high collinearity were removed.

Exploratory Data Analysis

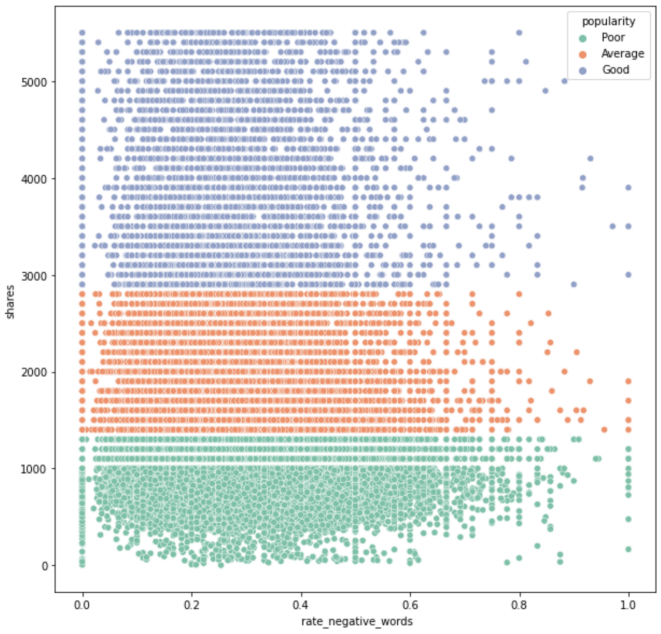
General distribution of the categories created based on threshold

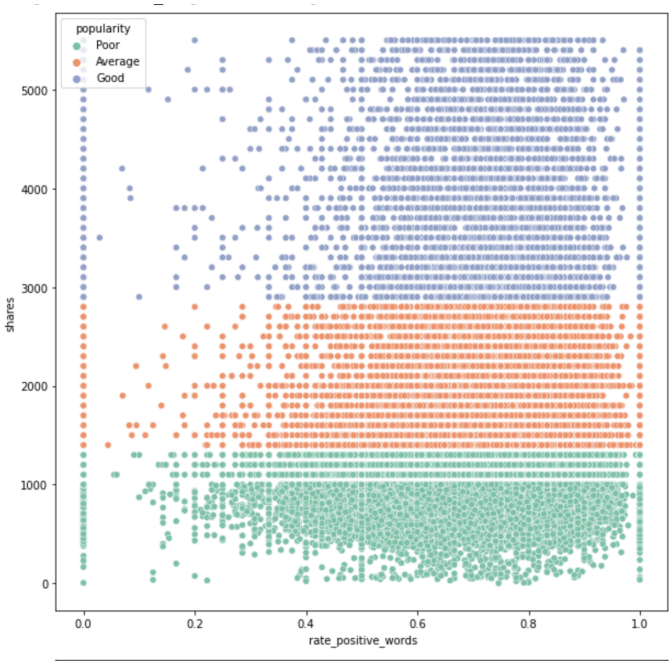


On plotting ‘No of Images vs. Popularity’, we see that the more the number of images in the article, the better the article performs and is popular.



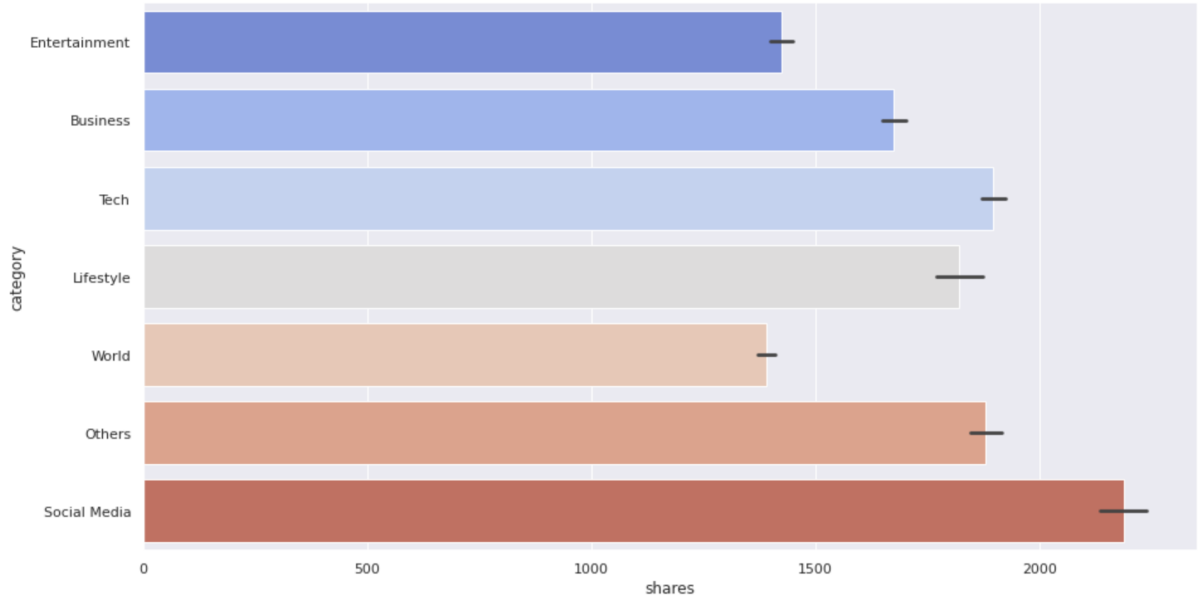
Looking at this graph we see that there are many outliers. We remove these outliers by checking the number of shares that are outside the lower and upper bounds, setting those values to null, and then dropping them. This improves the rest of our visualizations greatly.



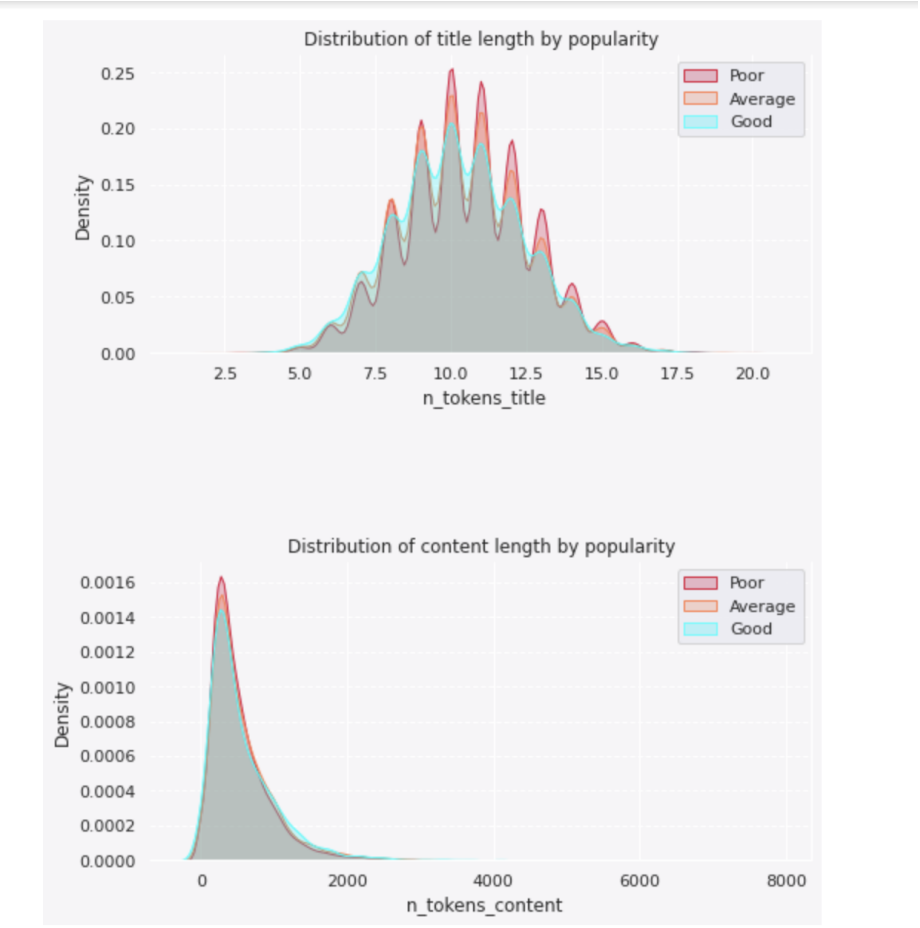


Rate of positive and negative words vs. shares. We see that although there is not much of a difference between the rate of positive and negative words and number of shares, articles seem to generally have a higher rate of positive words than negative words.

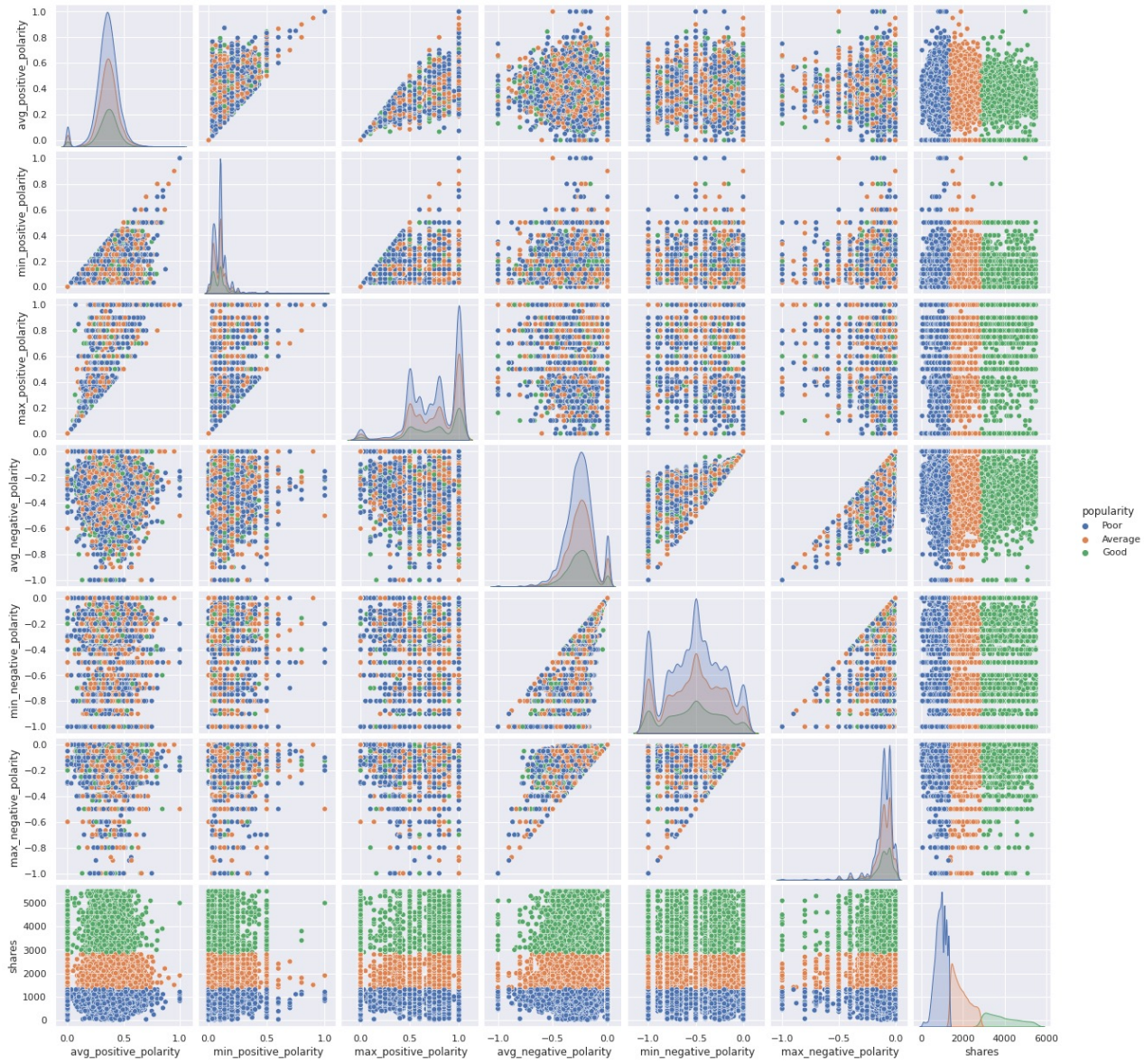
Popularity based on Category



By merging different category columns like lifestyle, entertainment, business, tech, etc. into one Category, we see that news articles based on social media are the most popular.



Although we might assume that title length and content length would affect the number of shares, we see that for each category, the distribution remains the same



We plot a pair plot for a few of the features to see the general trend among the dataset features.

Feature Selection

Feature selection techniques can avoid the curse of dimensionality and thus enable the simplification of models, making interpreting experimental results easier for researchers.

Mutual Information

The mutual information quantifies the "amount of information" obtained about one random variable by observing the other random variable. High mutual information indicates a large reduction in uncertainty; low mutual information indicates a small reduction; and zero mutual information between two random variables means the variables are independent.

	Variable	Mutual Information
0	n_tokens_title	0.325101
3	num_hrefs	0.317450
50	min_negative_polarity	0.307814
51	max_negative_polarity	0.295994
4	num_self_hrefs	0.283098
47	min_positive_polarity	0.271468
48	max_positive_polarity	0.269957
8	num_keywords	0.258085
5	num_imgs	0.234605
52	title_subjectivity	0.179812
54	abs_title_subjectivity	0.172882
37	LDA_02	0.169818
6	num_videos	0.162239
55	abs_title_sentiment_polarity	0.156902
36	LDA_01	0.151440
53	title_sentiment_polarity	0.146868
35	LDA_00	0.146592
24	self_reference_min_shares	0.138183
39	LDA_04	0.132762
38	LDA_03	0.131888

According to the table, we can see that `*LDA_03*`, `*LDA_04*` and `*self_reference_min_shares*` have the lowest scores, meaning they are more likely to be independent to the target variable.

Fisher Scores

As one of the supervised feature selection methods, the Fisher score algorithm selects each feature independently in accordance with their scores.

The key idea of Fisher score is to find a subset of features, such that in the data space spanned by the selected features, the distances between data points in different classes are as large as possible, while the distances between data points in the same class are as small as possible.

	Variable	fisher score
42	global_rate_positive_words	1.262845e+06
43	global_rate_negative_words	6.284018e+05
41	global_sentiment_polarity	4.497190e+04
40	global_subjectivity	2.287332e+04
37	LDA_02	2.071762e+04
47	min_positive_polarity	1.817810e+04
45	rate_negative_words	1.712397e+04
12	data_channel_is_socmed	1.556736e+04
32	weekday_is_saturday	1.312369e+04
34	is_weekend	1.184854e+04
36	LDA_01	1.102825e+04
14	data_channel_is_world	9.826235e+03
39	LDA_04	8.360767e+03
33	weekday_is_sunday	7.863596e+03
10	data_channel_is_entertainment	7.196600e+03
44	rate_positive_words	6.901474e+03
35	LDA_00	6.526622e+03
2	n_non_stop_unique_tokens	5.824602e+03
13	data_channel_is_tech	5.271595e+03
46	avg_positive_polarity	3.591342e+03

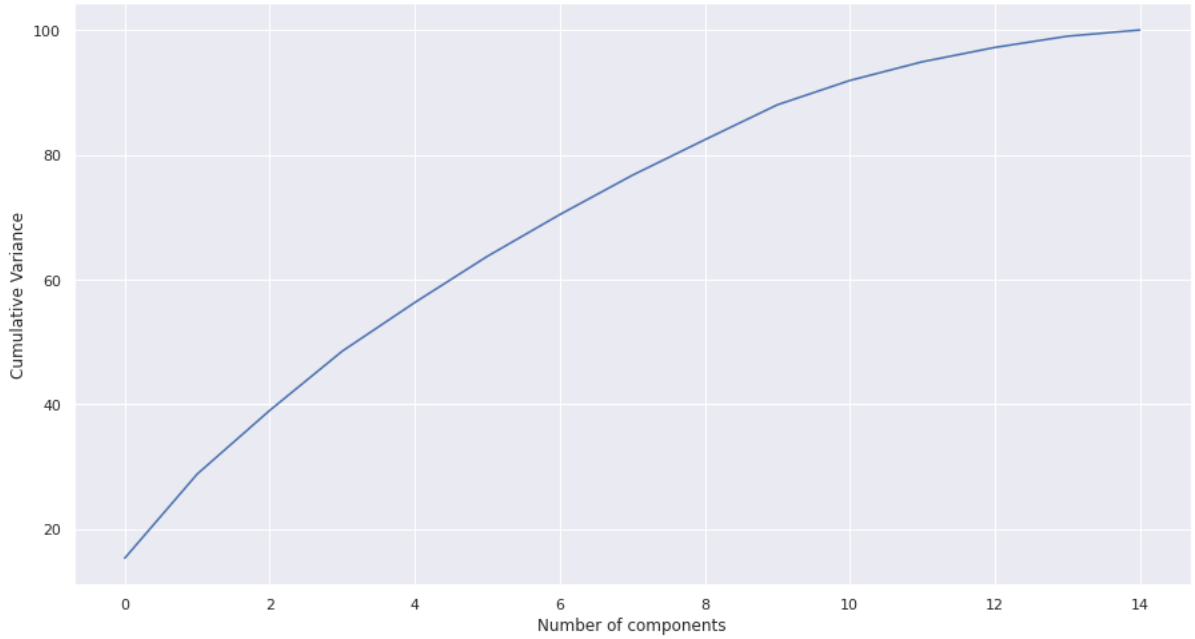
We can observe that `global_rate_positive_words` has the highest Fisher score.

	Variable	fisher score	Mutual Information
0	LDA_02	20717.6236	0.169818
1	min_positive_polarity	18178.0989	0.271468
2	LDA_01	11028.2498	0.151440
3	LDA_04	8360.7671	0.132762
4	LDA_00	6526.6218	0.146592

We select features based on the results of mutual information and fisher scores. The variables picked include common features among the two as well as the top 5 features from each of the two results that are not common.

Dimensionality Reduction

Principal Components Analysis



	Eigen Values	Eigen Vectors	Cumulative Variance (%)
0	2.301397	[0.31597307849334816, -0.0392869385279631, -0....	15.3
1	2.011386	[-0.1030293102416874, 4.225278730247548e-05, -...	28.8
2	1.540064	[0.13817908482280383, -0.4946146449461229, -0....	39.0
3	1.416273	[-0.1030079703713807, -0.07071042862060138, 0....	48.5
4	0.155994	[0.34108120089579236, 0.3320631293341278, -0.6...	56.3
5	0.269903	[0.35989341737448266, 0.2065851465772276, 0.18...	63.7
6	0.342201	[-0.5103002843013645, 0.296218917832869, 0.215...	70.4
7	0.441382	[0.23181009692852775, 0.04461440696423147, 0.3...	76.7
8	0.583675	[0.10702654885406943, -0.06813024680424386, 0....	82.4
9	1.178688	[-0.023136474171222218, 0.5798571338979243, -0...	88.0
10	1.102734	[-0.254127385371405, -0.08488694199096018, 0.0...	91.9
11	1.016055	[-0.08767599825856594, -0.08679313708429526, -...	94.9

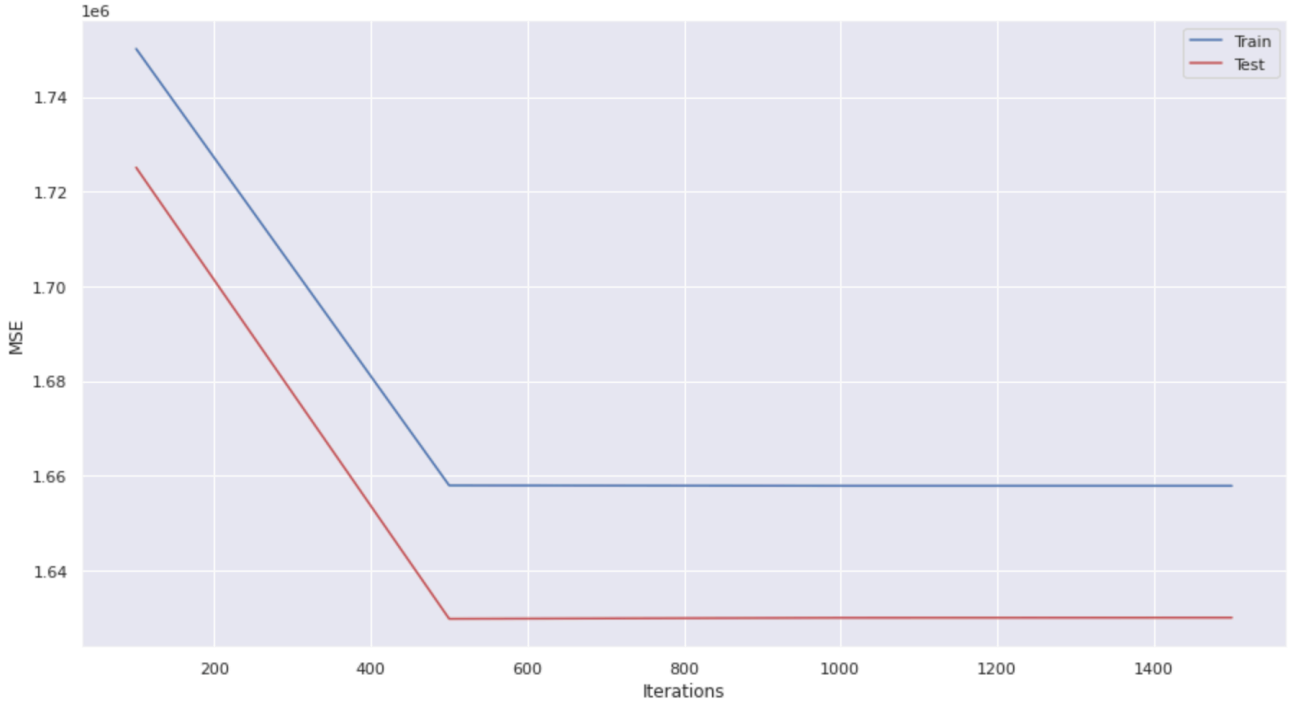
We observed that the cumulative variance reaches 95% when having 12 principal components. To preserve as much of the data's variation as possible, we chose the first 12 principal components for the prediction.

Model Implementation and Performance Evaluation

The regression models that have been used include Linear Regression, KNN, Decision Tree, Random Forest, and Neural Networks. Root Mean Squared Error and Mean Absolute Error will be used as the evaluation metric.

Linear Regression

As we have seen from some of the plots in our visualization section, although there is some linearity to the dataset features, most of them are not linear. However, since we are implementing regression, we are choosing to implement linear regression as it is the most basic form of regression. The learning rate is set at 0.01. To find the best epoch size we plot the MSE with a range of epoch sizes. From the plot below we see that there is no decrease in error after 500 epochs so we fit our model with that.



The algorithm used can be roughly described as follows:
1. We have a hypothetical function h(x) = weight*x + bias and a cost function defined as:

$$\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2$$

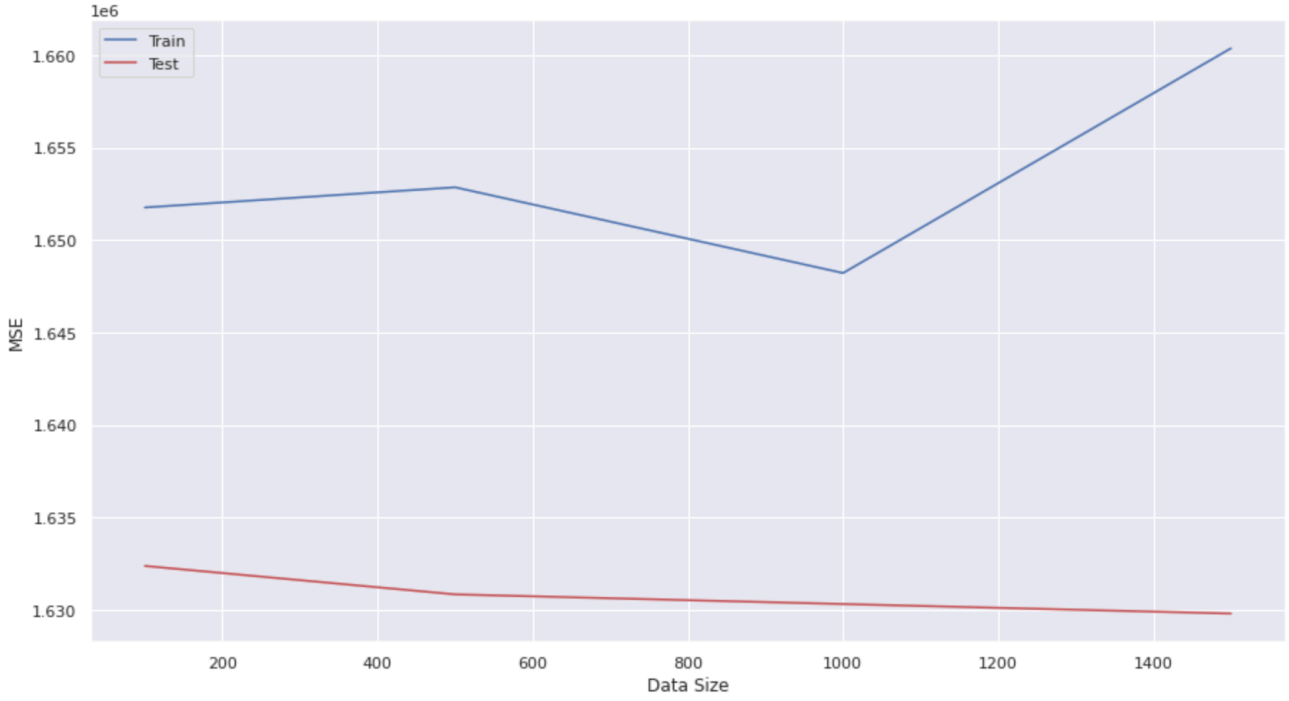
$$\frac{\partial}{\partial W}J = -\frac{2}{m}\Sigma_{i=1}^{m}(y_i - h(x_i)) * x_i$$

$$\frac{\partial}{\partial b}J = -\frac{2}{m}\Sigma_{i=1}^{m}(y_i - h(x_i))$$

dW and dB serve as the derivative of the loss function with regards to the weights and biases

2. The objective is to minimize this cost function, we will use gradient descent. To find the weight of any given feature we find a temporary weight temp = wi - learning_rate * dW and set it as the new weight for a given number of iterations (we found the number of iterations required for the

model in the plot above). The same is done for bias. Using the hypothetical function mentioned we can find the predicted values.
Implementing the model gives us the following results:
Time Taken: 2.28
RMSE 1282.21
MAE 1095.41



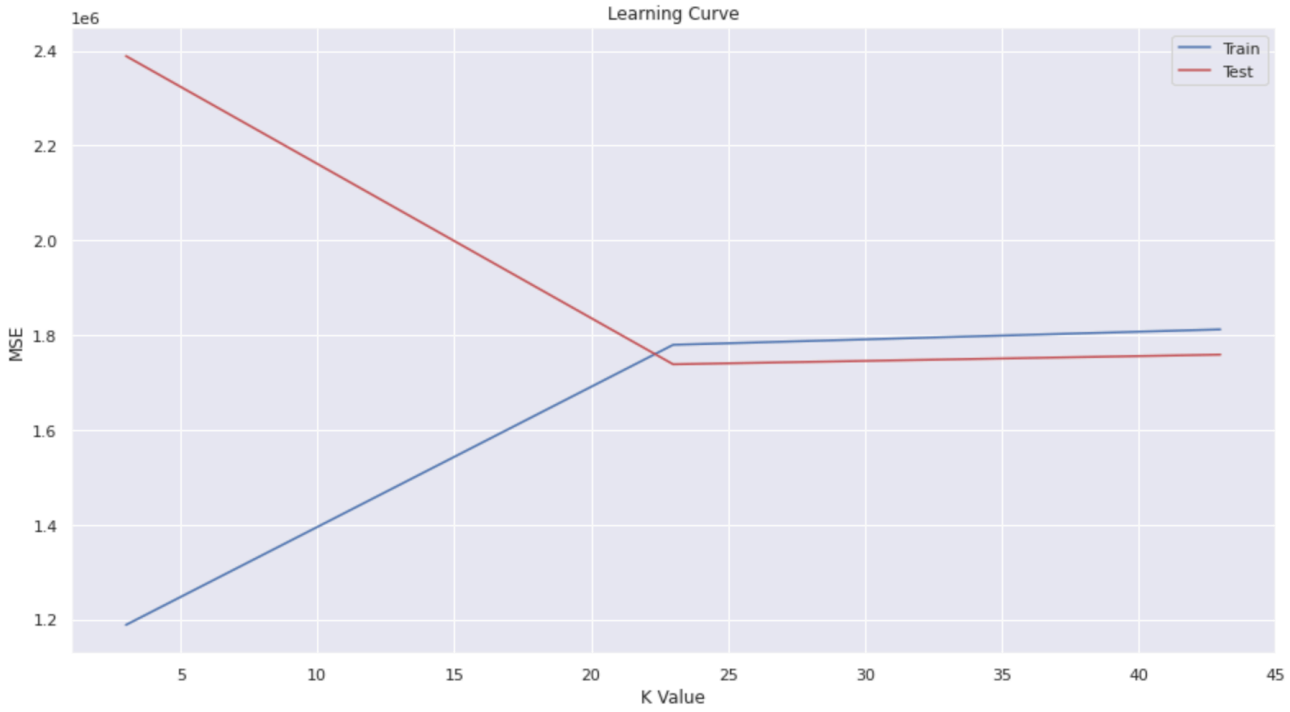
Using PCA
RMSE 1284.14
MAE 1097.53
Looking at the learning curve graph we can see why linear regression is not an ideal fit for our model, the high training data size as well as the non linearity of the model tends to generalize the data.

KNN

For a given value using the regression technique in the K-Nearest algorithm, instead of finding the unseen data points of classes of the K neighbors we take the value of the target. We find the target value for the unseen data points by taking the average.
For distance metrics, we will use the Euclidean metric.

$$d(x, x') = \sqrt{(x_1 - x_1')^2 + \ldots + (x_n - x_n')^2}$$

The final predicted value will be an average of the k nearest neighbors. Due to the time and memory used up by KNN we fit the model on a sample dataset. To find the optimal k value we plot k - value vs MSE. Looking at the plot we find the optimal k to be around 23 which is what we will use for implementation. We can also note that for the sample MSE for the train and test set almost converge



which is the ideal learning curve, however the inability of knn to handle data as large as our dataset makes it less feasible.
Training Time: 12.81
RMSE 1318.45
MAE 1135.53

PCA
RMSE 1418.94
MAE 1191.34

Decision Tree

Decision Trees works by using a top down approach. At each node the best split is determined by finding the variable that provides the highest information gain. Since we are dealing with regression we need to build a regression tree that works with continuous response variables. After the decision tree is constructed, the final result for any given test value will be an average of the target values at the leaf node. The algorithm used works as follows:

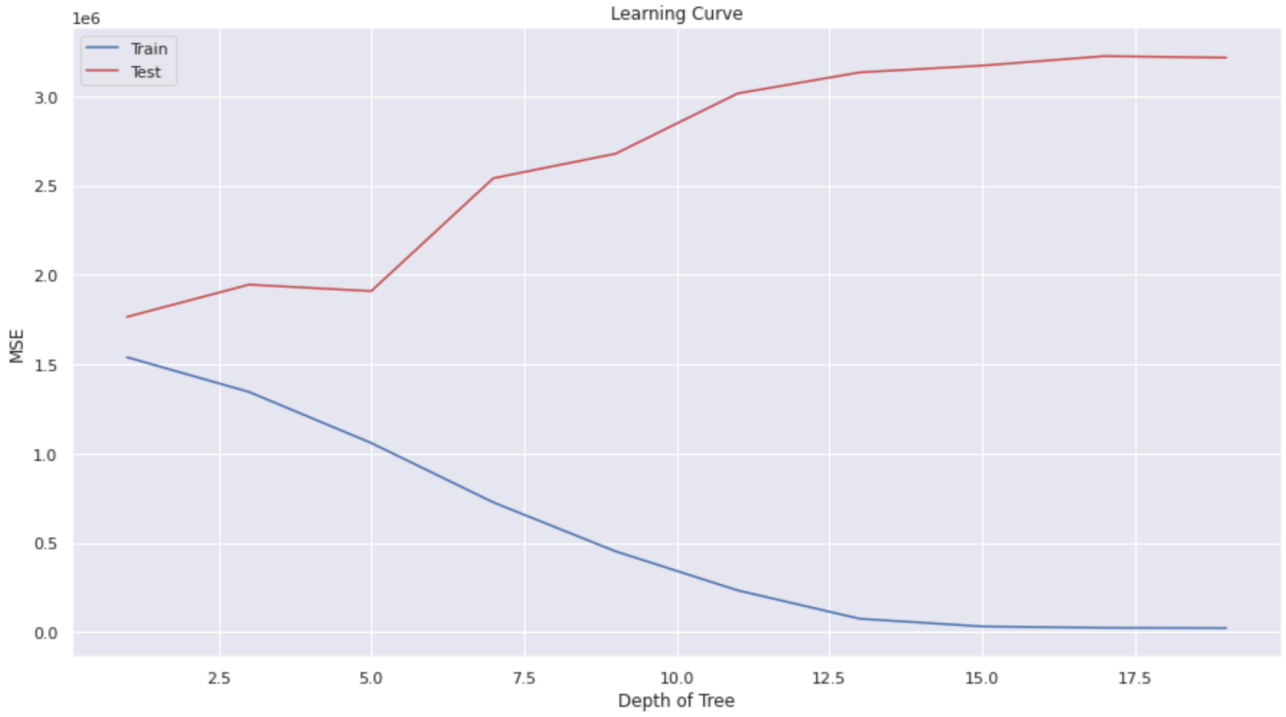
1. Define Node class: a constructor that contains variables that are required for the decision and leaf nodes. For each decision node: a.) it contains a record of which feature is acting as the splitting condition and the threshold corresponding to that feature b.) the left and right nodes to traverse from the parent node to the child nodes c.) variance reduction cost by a splitting corresponding to the decision node

2. Define Tree class: defines stopping criteria, min_sample_split and max_depth. Unlike in classification where we use gini index or entropy, for regression we will need to calculate variance. Reduction in variance is what is going to serve as an equivalent to information gain. The get best split function is another recursive function that gives us the best split based on the threshold for every decision node, finally if the variance reducing by best split is greater than 0

we split the tree. The leaf value at the max depth of every split is the average of the target values that satisfy the split conditions.

```
def variance_reduction(self, parent, l_child, r_child):
    weight_l = len(l_child) / len(parent)
    weight_r = len(r_child) / len(parent)
    reduction = np.var(parent) - (weight_l * np.var(l_child) + weight_r * np.var(r_child))
    return reduction
```

From plotting the MSE for a range of tree depths we see that as depth increases, decision tree tends to overfit the training data. To reduce the variance between the two predictions we pick a depth that does not over fit the data nor increase test error too much (5)



Time Taken: 14.46
RMSE: 1340.04
MAE: 1117.10

Using PCA values for the same depth:
Time Taken:5.67
RMSE: 1810.68
MAE: 1462.71
It can be noted that the MAE and RMSE values are slightly higher but the time taken is significantly lesser. The tree generated is also different, it can be compared with the tree generated without using PCA as follows:

```
X_0 <= 1.6016909302835767 ? 112888.41425424209
 left:X_14 <= 0.5931287122833061 ? 64535.22439102852
  left:X_11 <= -0.33528747216688015 ? 41211.34947318863
    left:X_8 <= -1.2912031807777382 ? 59240.76090503507
        left:X_14 <= -3.281413513845672 ? 120294.0162167106
                left:1900.8281734033185
                right:X_4 <= -0.6155192171180093 ? 10902.722222222223
                                left:983.5
                                right:762.0
        right:X_2 <= -0.5354845470779767 ? 56505.50670745503
                left:X_3 <= 2.366807581436305 ? 315378.14292030595
                                left:1782.6908111593464
                                right:5200.0
                right:X_13 <= -1.0334009800759734 ? 65662.90760813211
                                left:1818.4258151969036
                                right:2584.9047104666524
    right:X_1 <= 0.088567819305305 ? 98062.35424911743
        left:X_8 <= 0.8098662465920818 ? 169902.70012475457
                left:X_7 <= -1.082235348127149 ? 210955.75707007456
                                left:1220.4454151857276
                                right:3340.29648975821
                right:X_3 <= -0.22173073474307733 ? 1179793.8700291729
                                left:1621.7059633542297
                                right:5400.0
        right:X_8 <= 0.5226171014846656 ? 154541.90160959912
                left:X_14 <= 0.07766052521538165 ? 167841.95549269766
                                left:1915.4954893027982
                                right:2841.930984602549
                right:X_1 <= 0.08968681531610802 ? 490524.53082799073
                                left:2301.4028848028584
                                right:3702.5688417335687
  right:X_14 <= 0.641135738721983 ? 111361.81048712484
    left:X_13 <= 1.539953157854481 ? 86846.06350419472
        left:X_2 <= 1.775480528506638 ? 59157.15648638026
                left:X_5 <= 2.3197602058514795 ? 46978.8528628865
                                left:1734.5635669673532
                                right:3779.473807445262
                right:X_10 <= 0.32663562026565157 ? 124218.10013608813
                                left:752.875
                                right:1477.3206315521354
```

Without PCA

```
X_9 <= 0.6522413760515207 ? 105574.50173298945
 left:X_7 <= 0.8480813375197307 ? 83837.9683280387
  left:X_11 <= -0.7022000515500251 ? 92105.73132152553
    left:X_8 <= -1.1819709947074095 ? 426475.86826555885
        left:X_0 <= -0.8020293094996908 ? 810000.0
                left:4600.0
                right:2800.0
        right:X_6 <= 0.0471640042485867 ? 101666.21506872942
                left:X_1 <= -1.2285498364644138 ? 33691.50340136056
                                left:744.6666666666666
                                right:1192.0
                right:X_11 <= -1.090239715980358 ? 92342.37377781628
                                left:1366.6666666666667
                                right:1994.3561823658558
   right:X_5 <= 0.9554055948065328 ? 129137.50972164469
        left:X_3 <= 0.6421852680922507 ? 91369.28957925574
                left:X_9 <= 0.2495174272358287 ? 163495.59566241782
                                left:2878.219823971399
                                right:1876.2960245099491
                right:X_9 <= 0.16744803604700061 ? 222667.82216893637
                                left:1793.3320576592298
                                right:3270.3219842593667
        right:X_3 <= -1.284829160027611 ? 756581.2420571997
                left:X_0 <= -3.253557434993815 ? 82211.01631486283
                                left:3547.331087505013
                                right:4120.780357501454
                right:X_10 <= -0.3498254785045307 ? 87078.37868149162
                                left:645.0
                                right:1312.2984067957698
 right:X_9 <= -0.19758417470045536 ? 236202.85746997013
   left:X_2 <= -1.6923730159272683 ? 230193.9667893396
        left:3300.0
        right:X_0 <= -1.6929421638402538 ? 32231.222563766205
                left:X_0 <= -2.560160314293244 ? 22500.0
                                left:1700.0
                                right:1400.0
                right:X_2 <= 0.6308019341225775 ? 23912.830321222784
                                left:791.7
                                right:1101.4047293787605
```

With PCA

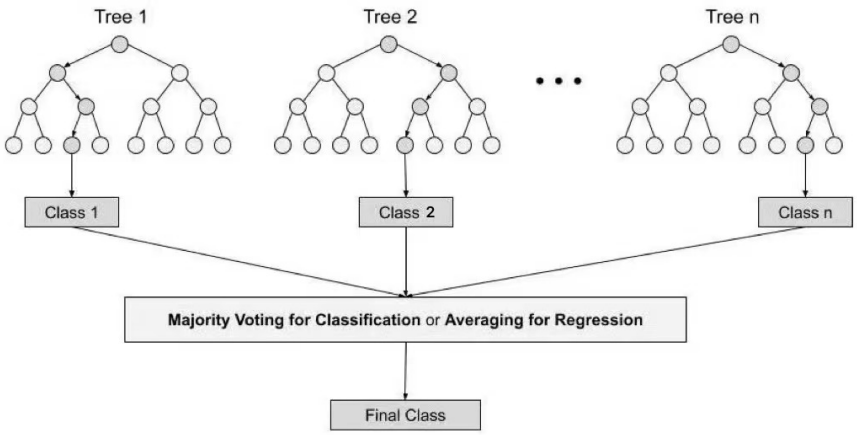
Random Forest

A random forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

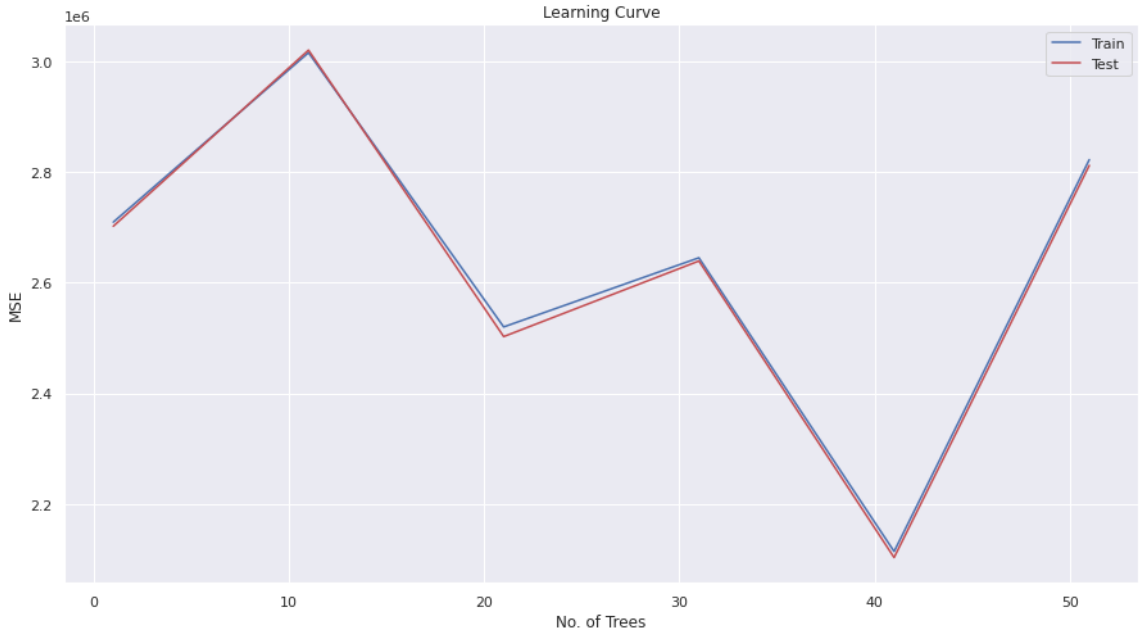
Within a random forest, there is no interaction between the individual trees. A random forest acts as an estimator algorithm that aggregates the result of many decision trees and then outputs the most optimal result.

The algorithm is as follows:

1. We have constructed basic decision tree model. For given number of trees, we took n number of random records using bootstrap method from the training data set.
2. Applied individual decision trees for each sample data set.
3. Each decision tree will generate an output.
4. Final output is considered based on ***Averaging*** for the regression.



Compared with Decision Trees, Random forests are created from subsets of data and the final output is based on average, and hence the problem of overfitting is taken care of. Random forest randomly selects observations, builds a decision tree and the average result is taken. It doesn't use any set of formulas. However, it is comparatively slower.



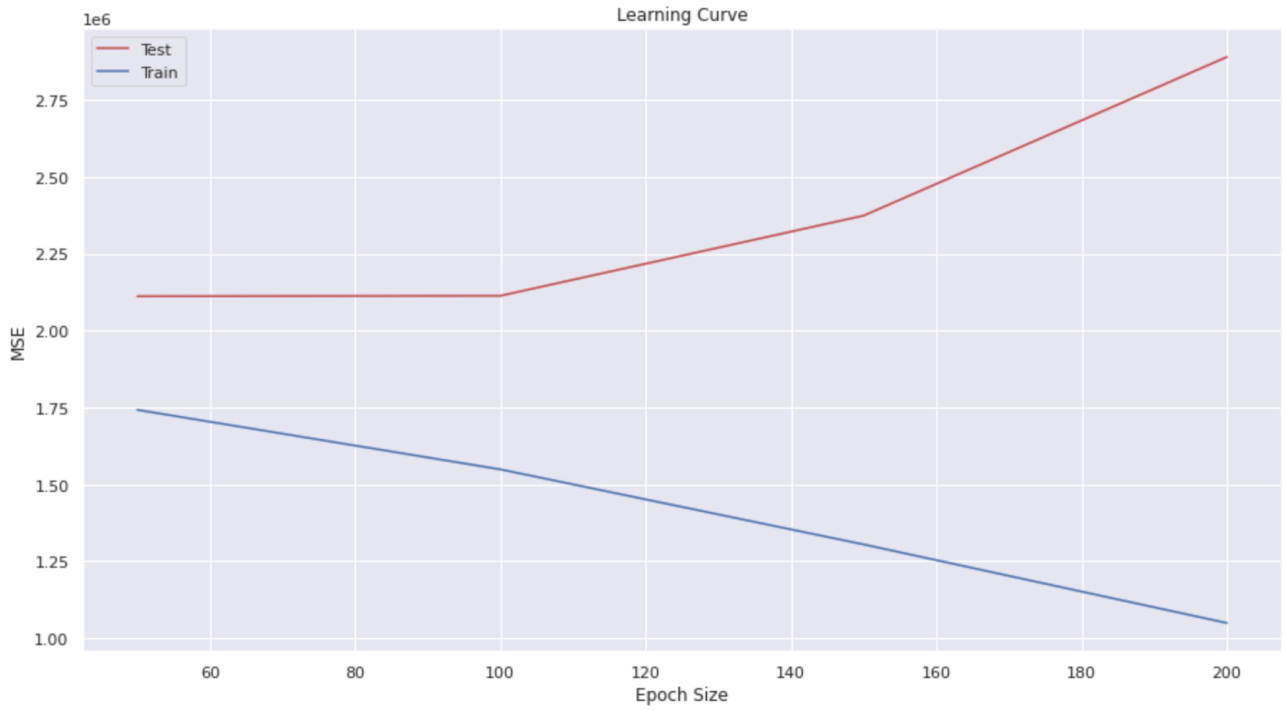
We observed that when having 41 trees in the random forest, the model has the minimum MSE.

Therefore, we built the random forest with 41 trees, the RMSE and MAE of the output are as follows:
RMSE 1384.98
MAE 1114.03

Using PCA:
RMSE 1458.26
MAE 1235.12

Neural Network

Tensorflow and Keras were used to construct a neural network using the Sequential API. For trying out different model we established a baseline batch size and epoch size. The batch size was set to 20, a smaller batch size generally gives lower errors although it has a longer execution time. The epoch size was determined by plotting the MSE of a sample size training and test data with a batch size of 20. On testing a sample dataset with different epochs, we see that with an increase in epoch size the training error reduces exponentially, while the testing error starts increasing after 100 epochs, thus indicating overfitting. So for the different models were training with a batch size of 20 and epoch size of 100, and switching the number of hidden layers used.



Five different models were tested and based on the results we chose model 4 that had 5 hidden layers, activation function 'relu', and the Adam optimizer. Model 4 performs better than the rest. Time Taken: 251.21s ()

	No. of Hidden Layers	RMSE	MAE
Model 1	2	1266.27	993.48
Model 2	3	1269.03	999.66
Model 3	4	1265.06	982.67
Model 4	**5**	**1265.36**	**972.08**
Model 5	6	1284.92	984.51

Results

Model Comparison

Model	RMSE	MAE
Linear Regression	1276.66	1089.08
KNN	1318.45	1135.53
Decision Tree	1340.04	1117.10
Random Forest	1384.98	1114.03
DNN	1284.92	984.51

Conclusion

- We can conclude based on our results that for our dataset, Neural Network and Random Forest have the best overall performance
- The execution time for Random Forest is notably lesser than Neural Network, as it takes a few seconds as opposed to several minutes for DNN
- Our KNN model for the sampled data shows the most ideal learning curve, with low overfitting (low variance). However, it is not a feasible model to implement considering the size of our dataset
- Random Forest and Neural Networks algorithms also allow for more tuning due to the number of hyper parameters that can be changed
- It can be noted that by selecting different features, or by taking different percentage of features than what has been used for our models, the performance could be improved