SDA PROJECT

06.12.2020

Contents:

Group 22:

Rukmini Meda (S20180010102)

Aditi Verma (S20180010006)

Nethra Gunti (S20180010061

Isha Aggarwal (S20180010067)

Abstract

In this project, we aim at predicting and classifying the number of shares of online news articles. The classification is a binary classification problem where we have to label the article as either ‘popular’ or not based on the number of their shares. The threshold for classification is the number of shares greater than or equal to 1400. For this, we have fit the data into several models and compared the results. The dataset being used comes from Mashable, a well-known online news website.

Libraries

This task has been done in *Python 3.8* and following are the libraries used:

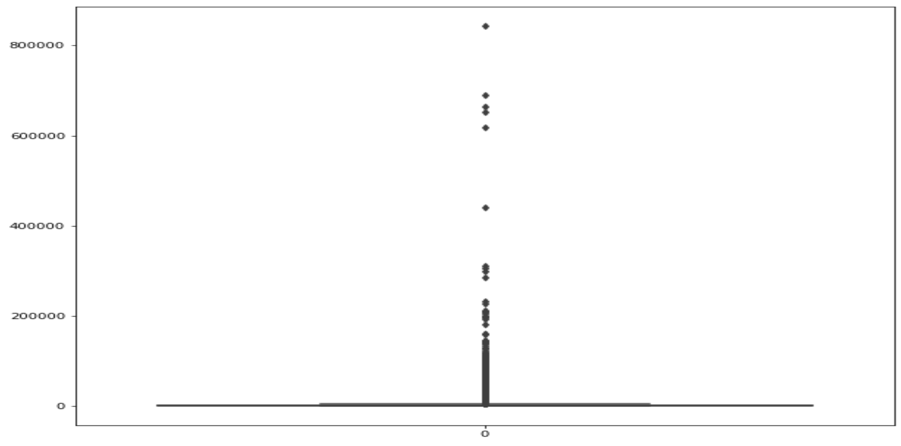
1. Numpy *1.18.1*
2. Pandas *1.1.4*
3. Sklearn *0.0*
4. Matplotlib *3.3.2*
5. Seaborn *0.11.0*
6. *Scipy*
7. *Statsmodels*

Exploratory Data Analysis (EDA)

Summary

Overall statistics

This dataset is obtained from mashable, a well known news site. It contains 39644 observations with 61 features. There are two non predictable features, url and timedelta, one target variable, shares among them. On observation, the data has feature labels with redundant spaces. Hence, the data is pre processed to remove redundant spaces and drop non predictable features from data. On observing the summary statistics, all the features are of float data type except for shares which is of int. Describing the data using pandas method showed that most of the variables had their maximum values pretty larger than their mean values. We can interpret that there can be outliers in the data. The boxplot of the entire dataset is plotted to observe the same.



Target Variable Summary

The target variable, shares has a mean of 3395.3801 with a minimum value of 1 for 17266th observation and a maximum value of 843300 for 9365th observation.

Descriptive statistical Analysis

As the dataset has more than one feature it is multivariate. It’s mean vector, covariance matrix and correlation matrix are computed to plot the heatmap.

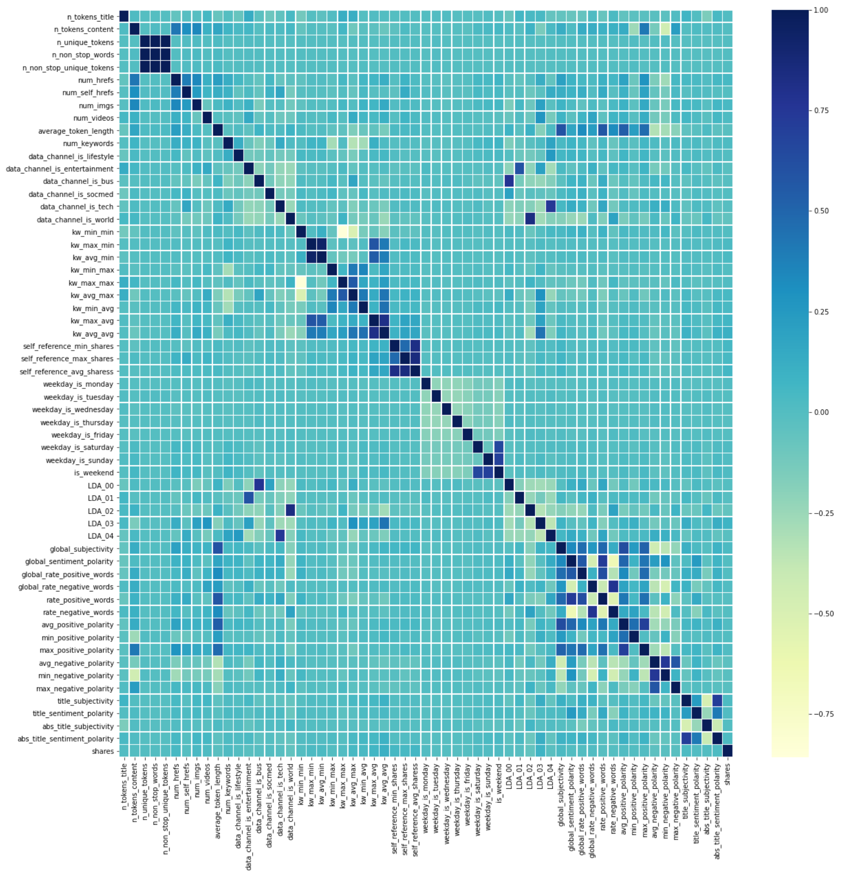


Fig. Heatmap of the correlation matrix computed

We can observe that a lot of features have very low correlation amongst them especially in the last column of the heatmap where correlation between shares and all other features is nearly zero. We can infer that the features nearly do not share a linear relationship.

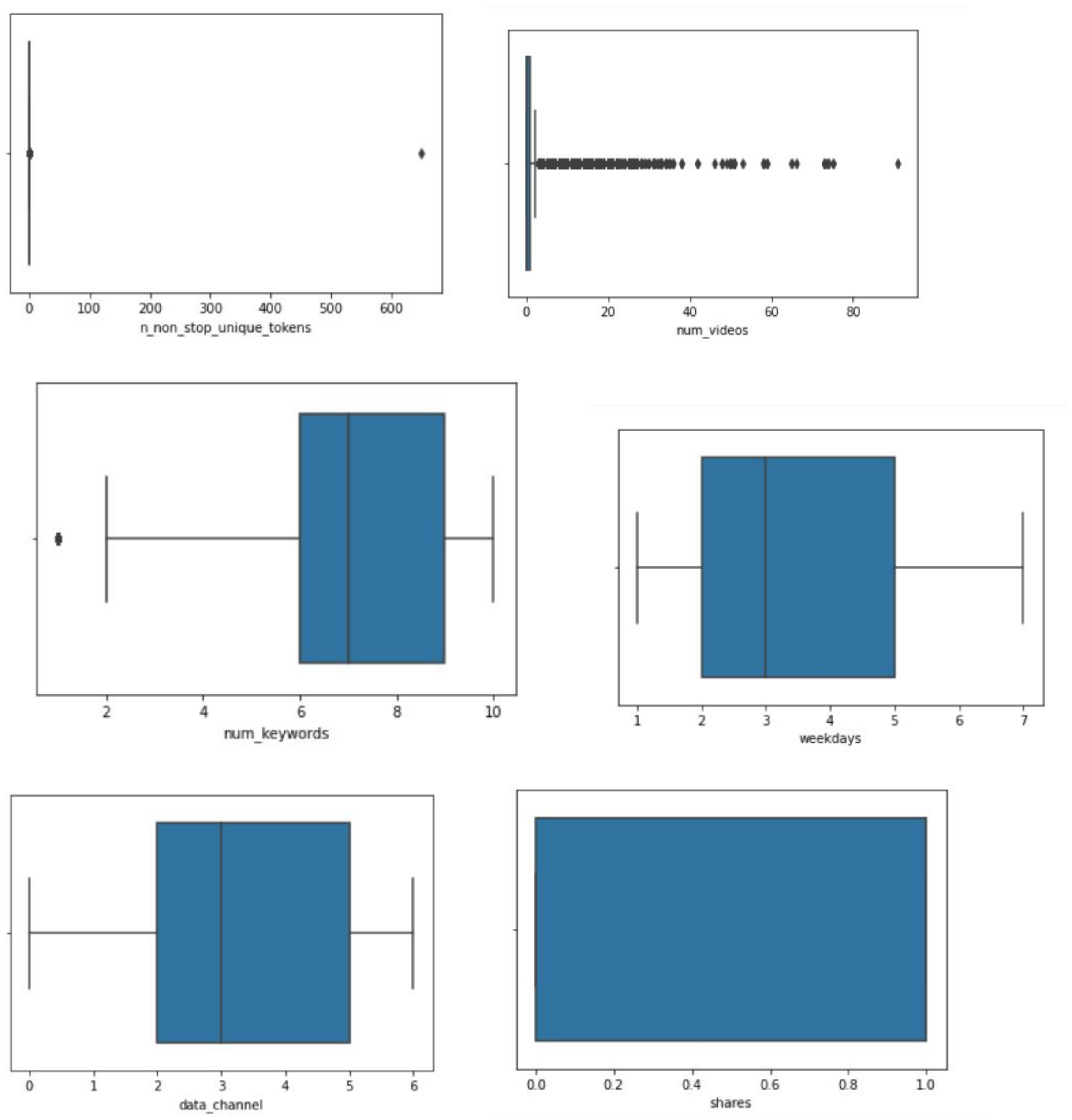


Fig. Boxplots of a few features, unique non stop tokens, number of videos, number of keywords, weekdays, data channel and shares.

From the above boxplots, we can observe that many features have high variance and outliers.

Noise Removal

Many features have many outliers and hence can be dropped from the dataset. One such feature, non stop words is dropped and individual features of weekdays and data channels are merged.

Normality Checks

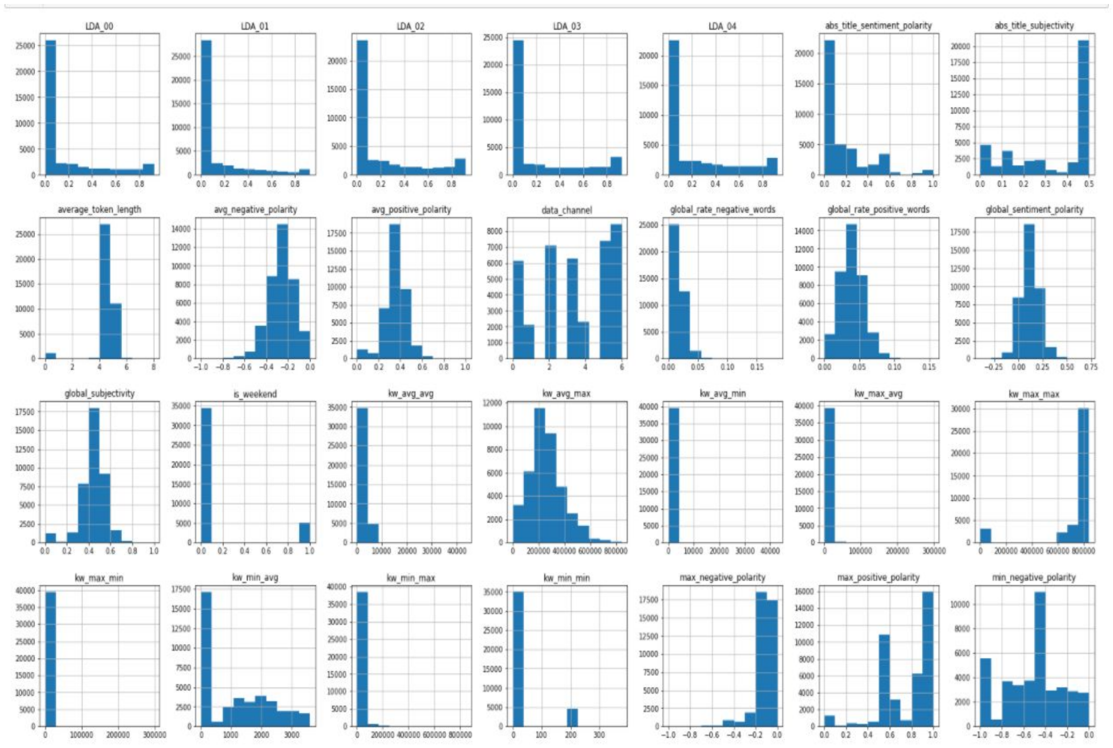


Fig. Histograms of a few features

Visual Checks

Understanding distribution of the dataset is helpful for deciding with the model. To check for normal distribution, visual and statistical checks are done. Histograms of various features are plotted to observe the bell shaped curve of normal distribution. We can find the shape for a few features.

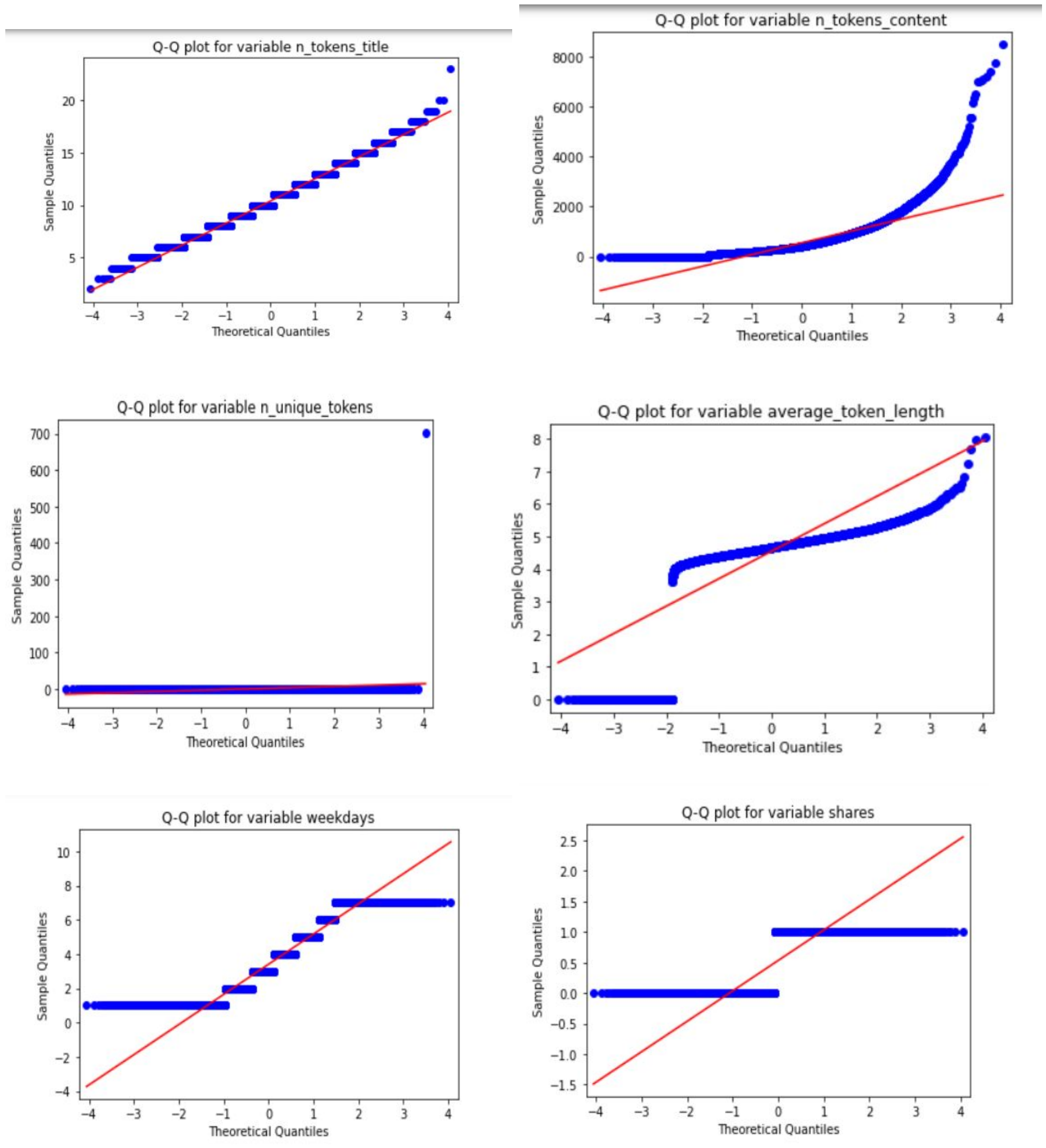


Fig. QQ-Plots of a few features

QQ Plots for a few features are shown above, we can find that most of the features do not belong to normal distribution including the target variable.

Statistical Check

We have performed the Shapiro Wilk Test and observed that the data does not belong to Normal distribution from the p-values.

Methodologies

Multiple Linear regression model and classification

First, we perform a test to check if the mlr model is suitable for the given dataset or not.

Test of Goodness of Fit

R_square = 1 - (SSE/SST) is found out along with **R_sqr_adj = 1 - (SSE/(n-p-1))/(SST/n-1).**

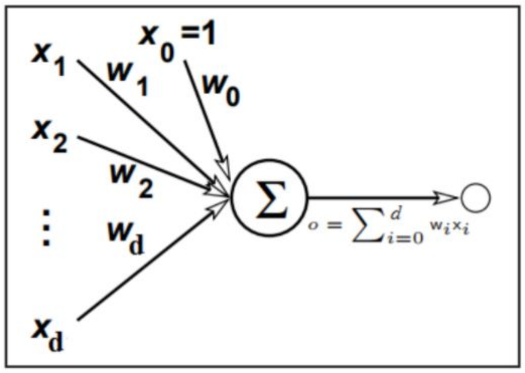
Note: R_square is the coefficient of determination.

If R_square >= 0.90 and R_sqr_adj >= 0.90, then the MLR model is really a good fit for the dataset.
We get:
R_sqr = 0.44714110507469407
R_sqr_adjusted = 0.44714114775479474
This is still fine, and we proceed to apply the MLR model.

MLR for Regression and Classification

Given a training dataset(data matrix and target variables), we try to fit a linear function of the random vector(from dependent variables to independent variable). Here the dependent variables include 58 predictive variables(all except url and time_delta) and the independent variable is the number of shares.

Y = beta_0 + beta_1 X1 + beta_2 X2 + ... + beta_p Xp + error (General MLR equation)



Error term is not shown in the above diagram.

Y is the independent variable while X1, X2, …., Xp are the dependent variables.

Beta_0, beta_1, ….. , beta_p are called the parameters or the regression coefficients- which are constant and unknown. Error term is random in nature. We aim to find an estimate- beta_i_hat for each of the regression coefficients.

<u>Note: We make certain assumptions before applying the mlr model</u>

(1) Linearity relationship : dependent variable as linear function of independent variables
(2) Uncorrelated residuals
(3) Normality of residuals
(4) Homoscedasticity

These assumptions would be further explained while dealing with tests of assumptions etc. We find estimates for optimal parameters (beta0_hat , beta1_hat , beta 2, … , beta_n_hat) using gradient descent method such that the sum of squared errors i.e. (½)* sum of (y_actual - Y_pred)^2 over the entire training dataset is minimized.

The cost function: J(beta_hat_vec) = (½)* sum over entire train set(Y_actual - Y_pred)^2

<u>Gradient descent step is:</u>
Let an augmented random vector from the train set be RV. (We accommodate beta_0_hat by augmenting 1 to the random vector in the 0th position).

Grad(cost_function) = sum over the examples considered(Y_actual - beta_hat_vec.RV)(- RV)

Beta_hat_new = Beta_hat_old + -eta*Grad(cost_function) (Eta is the learning rate)

Note: We implemented an online update procedure, where only 1 example is considered for an update of Beta_hat.

Once the optimal parameters(regression coefficients) estimates are found, for any augmented random vector RV, we can predict a Y value = beta_hat_cap.RV

actual_Y - predicted_Y is the estimate of the residual/error term.

The predicted value can further be used to classify the random vector. Here:
If predicted_Y >= 1400, class: “Popular”
Else class: “Unpopular”

Implementation: **All implementation is done from scratch using step by step descriptive functions for regression and classification as an extended outcome of it.**

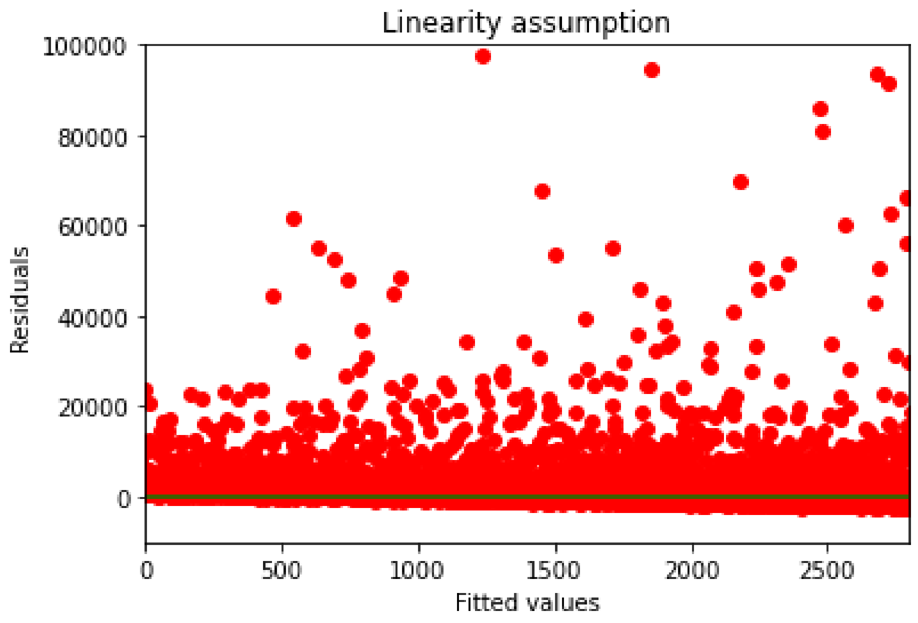
We achieve an accuracy of around 53% and 56% on test and train sets respectively.

Test of Assumptions

After, using the mlr model, we test the assumptions that we made in the beginning before applying mlr concepts.

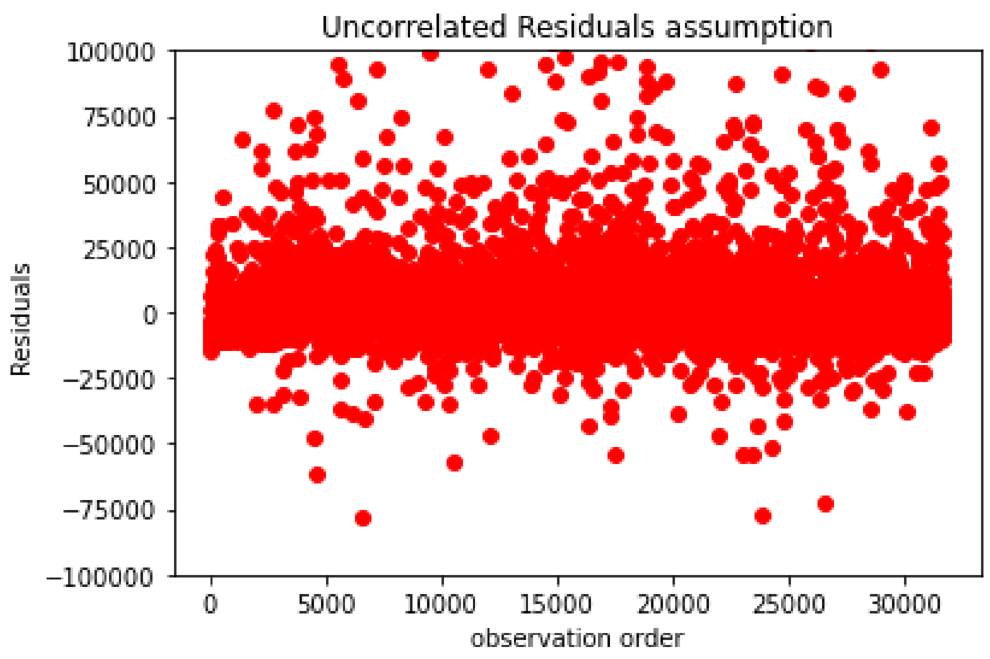
Test for Linearity Assumption

We assumed that the dependent variable is a linear function of the independent variables . This implies that the plot of residuals (Y-actual - y_predicted) vs Fitted values should not form a semicircular kind of ring around the y = 0 line. In the plot below we can see that there is no such pattern. Hence the linearity assumption is satisfied.



Test for Uncorrelated Errors Assumption

We assumed that the residuals for different observations are all uncorrelated. This implies that the plot of residuals vs ordered observation should not have a pattern, it must look random.



The above plot looks random in nature, hence we can say the assumption is satisfied.

Also, we perform a quantitative test: Durbin-Watson test for more accurate analysis.
DW is a statistic =

$$d = \frac{\sum_{t=2}^{T}(e_t - e_{t-1})^2}{\sum_{t=1}^{T} e_t^2}$$

Here T is the total number of observations, while e_t represents the residual estimate of t th observation.

If DW = 2, then no correlation
If DW >2, negative correlation
If DW < 2, positive correlation

Here we get DW ~ 1.99, which is quite good and means almost no correlation.

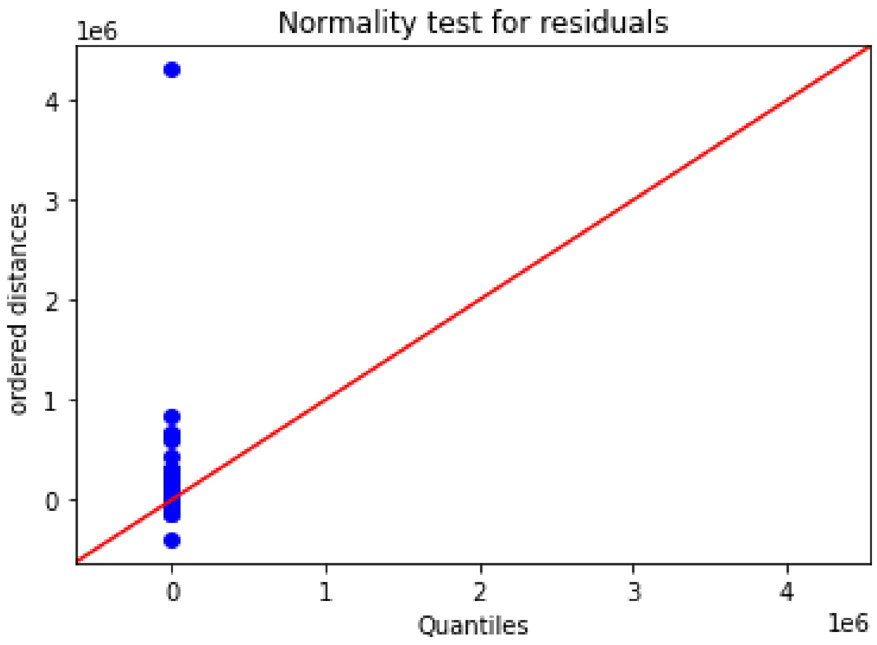
Hence, the assumption is satisfied.

Test for Normality of Residuals Assumption

We assumed that all residuals are normally distributed. To check this, we perform a Q-Q test.

We first find the sample mean (E_bar) and the sample covariance (S) and use these to find mahalanobis distance squared for each estimated residual.
d2_ i = (Ei_hat – E_bar).transpose * S.inverse* (Ei – E_bar) for the i th estimated residual value.

Now we plot: (d2 (1), q1) , (d2 (2), q2) , … , (d2 (n-1), qn-1) , (d2 (n), qn) , where d2 (j) is the jth number when squared mahalanobis distance is sorted in increasing order and qj is such that P(W >= qj) = (n – j + ½)/n when W ~ chi_square(1), n = number of estimated residuals = number of observations in the training set.



We get the above plot, this shows that the assumption is not satisfied, since the points are not located close to the 45 degree line

Test for Homoscedasticity of Residuals Assumption

When residuals do not have constant variance, it is difficult to determine the true standard deviation which usually results in confidence intervals that are too wide/narrow. In order to check if the residuals are homoscedastic, we look at a plot of residuals vs predicted values. If the residuals grow as a function of the predicted value, residuals might not be homoscedastic.

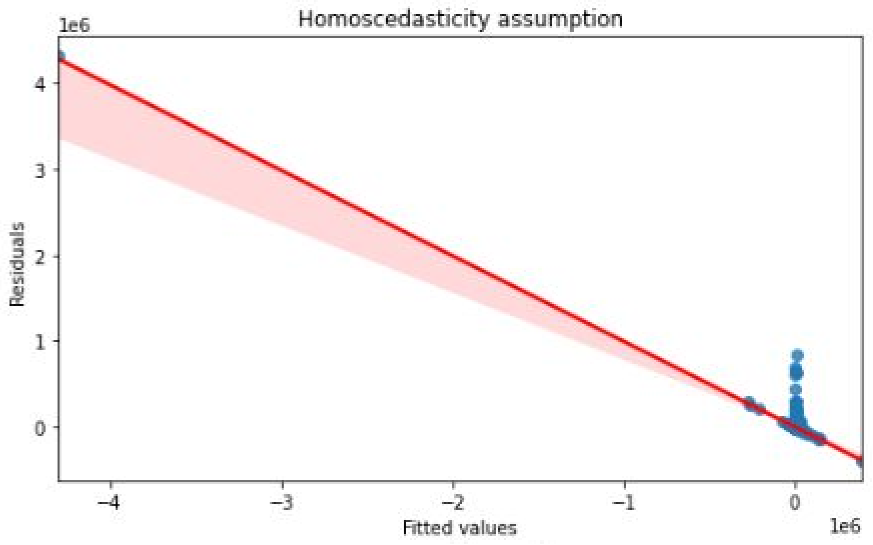


Fig. Regression plot (Predicted Values vs Residuals)

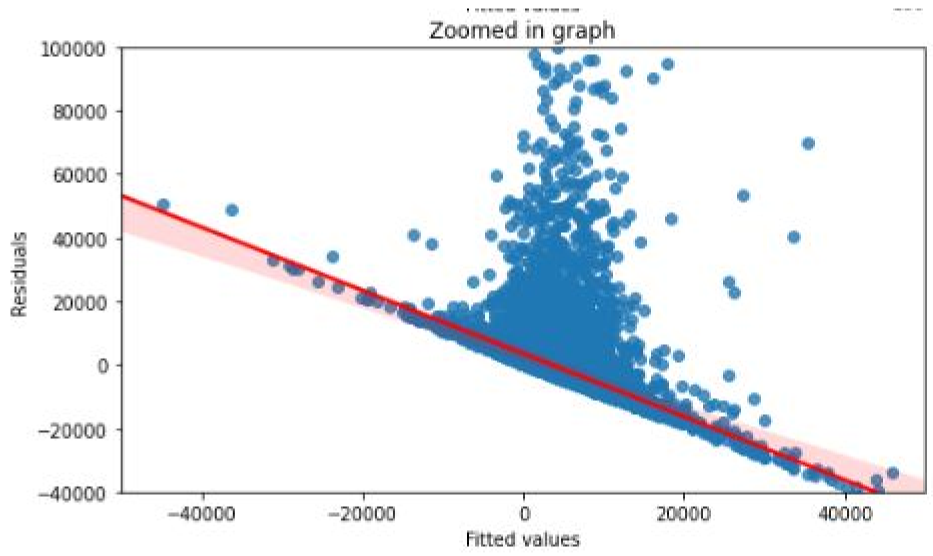
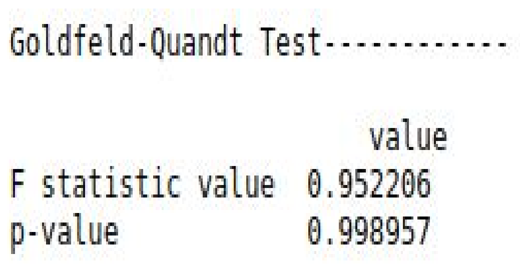


Fig. zoomed in graph of the above regression plot

This graph clearly shows that the data is indeed **Homoscedastic**. Inorder to confirm this, we also used a statistical test: **Goldfeld-Quandt**. In this test, the null hypothesis assumes homoscedasticity and a p-value below 0.05 indicates that the null hypothesis should be rejected in the favor of heteroscedasticity.



```
Goldfeld-Quandt Test-----------

                     value
F statistic value  0.952206
p-value            0.998957
```

Fig. Output of Goldfeld Quandt Test

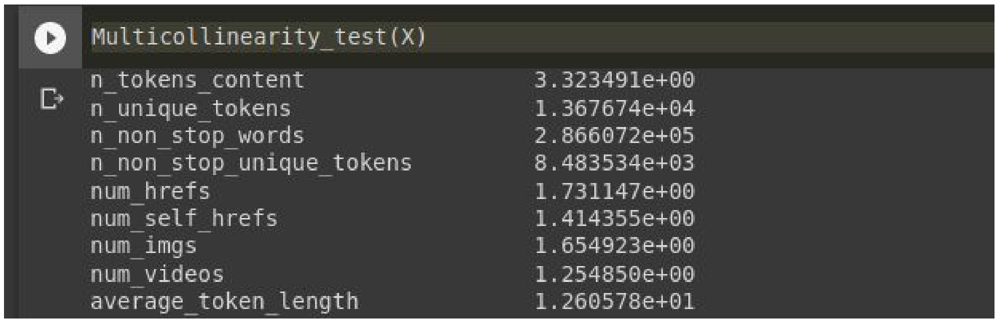
The p value is less than 0.05 which means that the Goldfeld Quandt test has not rejected the null hypothesis.

Hence we can say that the null hypothesis holds true and the data is Homoscedastic.

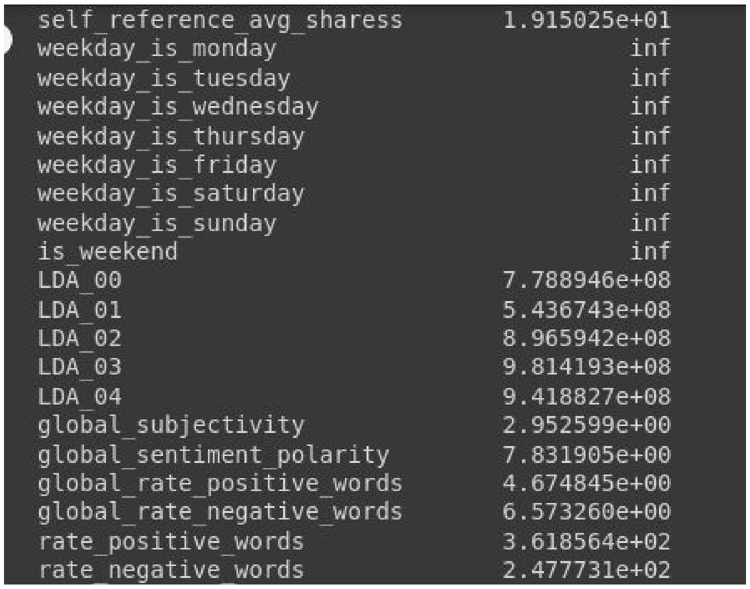
Diagnostic Tests

Multicollinearity Test

We perform a Multicollinearity test on the data.



```
Multicollinearity_test(X)
n_tokens_content                3.323491e+00
n_unique_tokens                 1.367674e+04
n_non_stop_words                2.866072e+05
n_non_stop_unique_tokens        8.483534e+03
num_hrefs                       1.731147e+00
num_self_hrefs                  1.414355e+00
num_imgs                        1.654923e+00
num_videos                      1.254850e+00
average_token_length            1.260578e+01
```



```
self_reference_avg_sharess        1.915025e+01
weekday_is_monday                          inf
weekday_is_tuesday                         inf
weekday_is_wednesday                       inf
weekday_is_thursday                        inf
weekday_is_friday                          inf
weekday_is_saturday                        inf
weekday_is_sunday                          inf
is_weekend                                 inf
LDA_00                            7.788946e+08
LDA_01                            5.436743e+08
LDA_02                            8.965942e+08
LDA_03                            9.814193e+08
LDA_04                            9.418827e+08
global_subjectivity               2.952599e+00
global_sentiment_polarity         7.831905e+00
global_rate_positive_words        4.674845e+00
global_rate_negative_words        6.573260e+00
rate_positive_words               3.618564e+02
rate_negative_words               2.477731e+02
```

Here we can see that some of the features have very high variance inflation factor (VIF), and thus high Multicollinearity and thus are redundant. They should not be considered.

n_unique_tokens

n_non_stop_words

n_non_stop_unique_tokens

weekday_is_monday

weekday_is_tuesday

weekday_is_wednesday

weekday_is_thursday

weekday_is_friday

weekday_is_saturday

weekday_is_sunday

is_weekend

LDA_00

LDA_01

LDA_02

LDA_03

LDA_04

In order to solve this problem, we perform principle Component Analysis

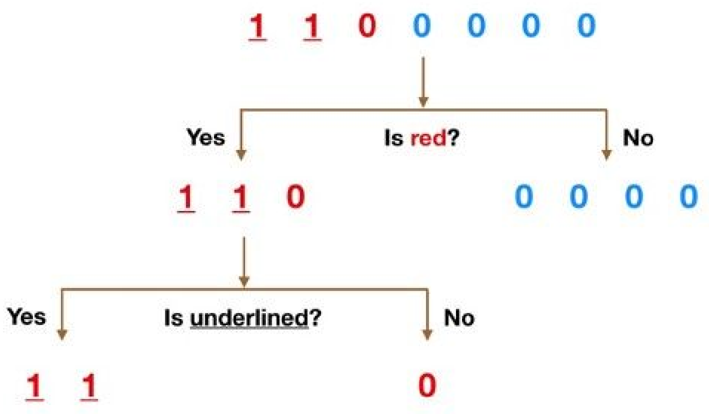
Other Machine Learning Approaches

We explored a few Machine Learning algorithms which could perhaps fit the given problem statement better than MLR. However, since this project was done from a statistical point of view (and not a Machine Learning one), we did not dive deep into these.

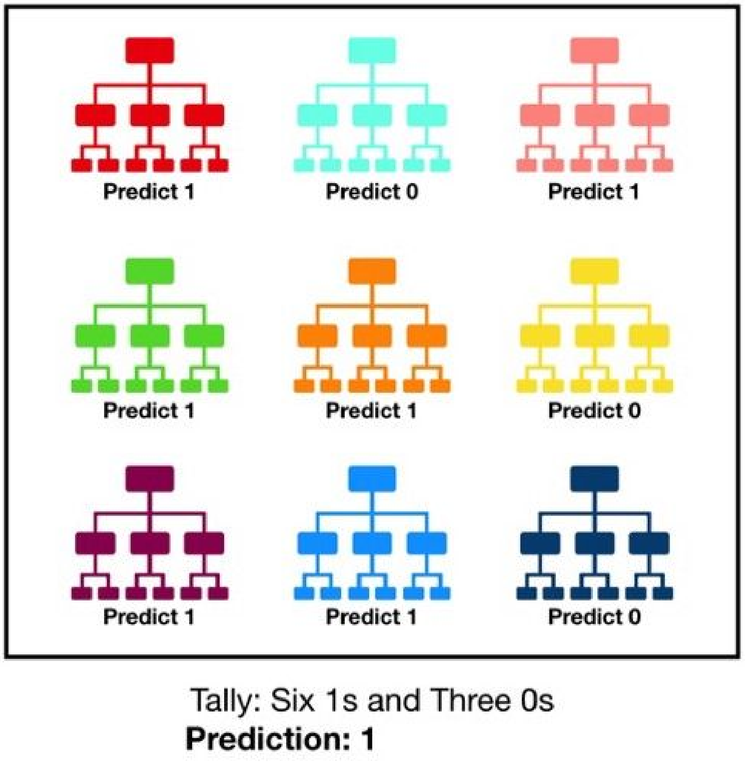
Random Forest

Here, we have used RandomForests, an ensemble Machine Learning algorithm, for the tasks of Regression (predicting the number of shares for a given article) and Classification (predicting whether the article will be popular, shares>1400). Random forests operate by constructing a multitude of decision trees (another type of machine learning algorithm) at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees. Random forests generally outperform decision trees.

Decision trees are an ML algorithm which divides every sample into categories, based on a set of conditions applied on its feature values, at each node. Each of the divided nodes is then further divided into child nodes based on a different set of assumptions. The predictions are made when each of the leaf nodes has a given number of samples.

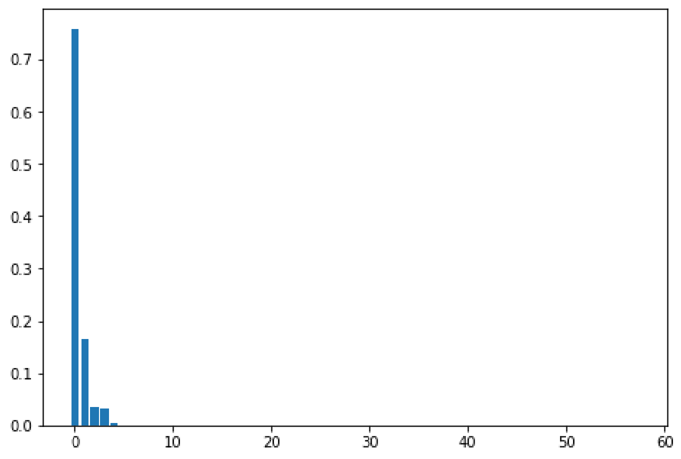


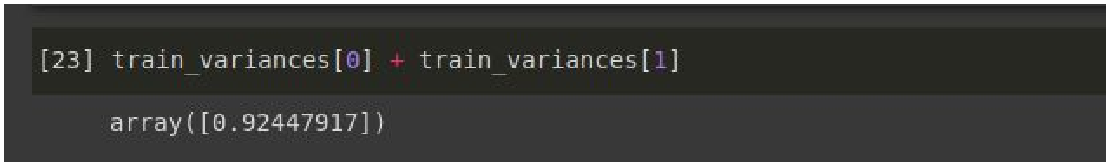
Random Forest takes a set of these decision trees, trains them on a set of **bagged samples**, and takes a soft majority of each of the trees results.



In this report, we have used the RandomForestRegressor and RandomForestClassifier models from the sklearn library.

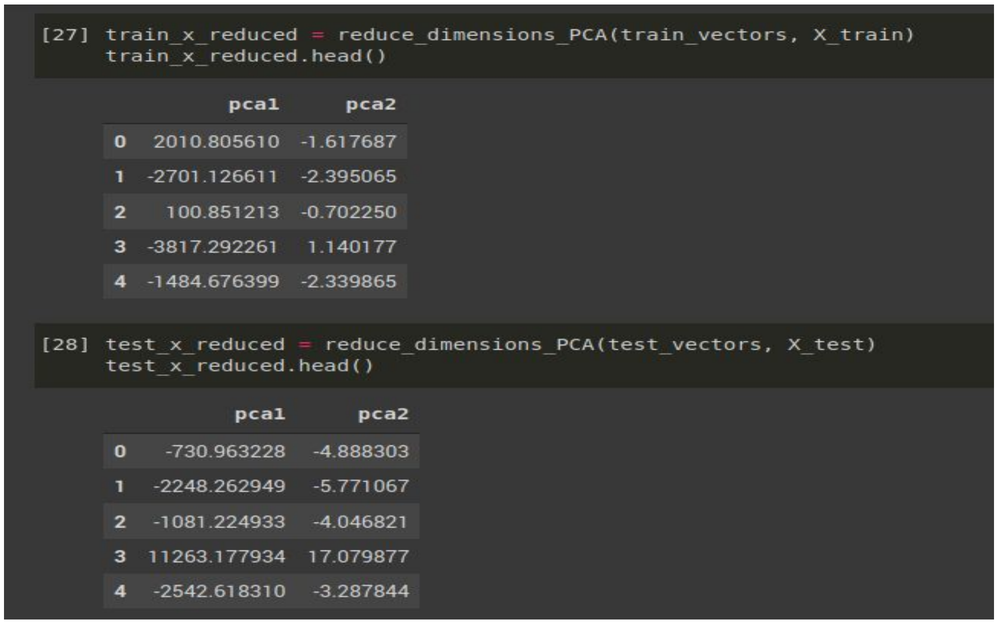
We perform PCA for the given data set and compare results between the original dataset and the reduced dataset.





```
[23] train_variances[0] + train_variances[1]

     array([0.92447917])
```

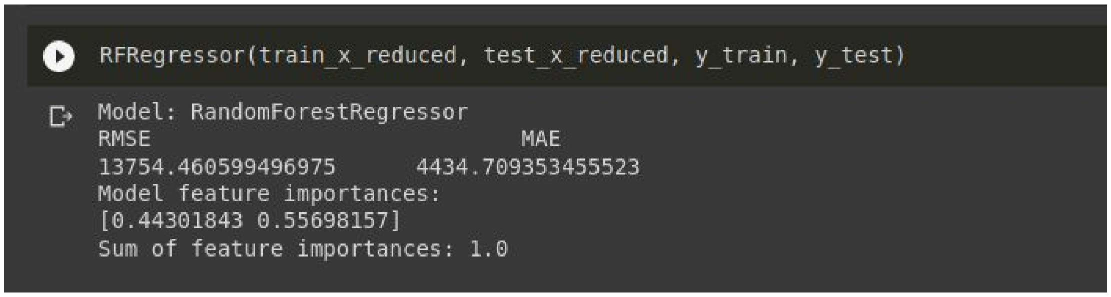


```
[27] train_x_reduced = reduce_dimensions_PCA(train_vectors, X_train)
     train_x_reduced.head()
```

	pca1	pca2
0	2010.805610	-1.617687
1	-2701.126611	-2.395065
2	100.851213	-0.702250
3	-3817.292261	1.140177
4	-1484.676399	-2.339865

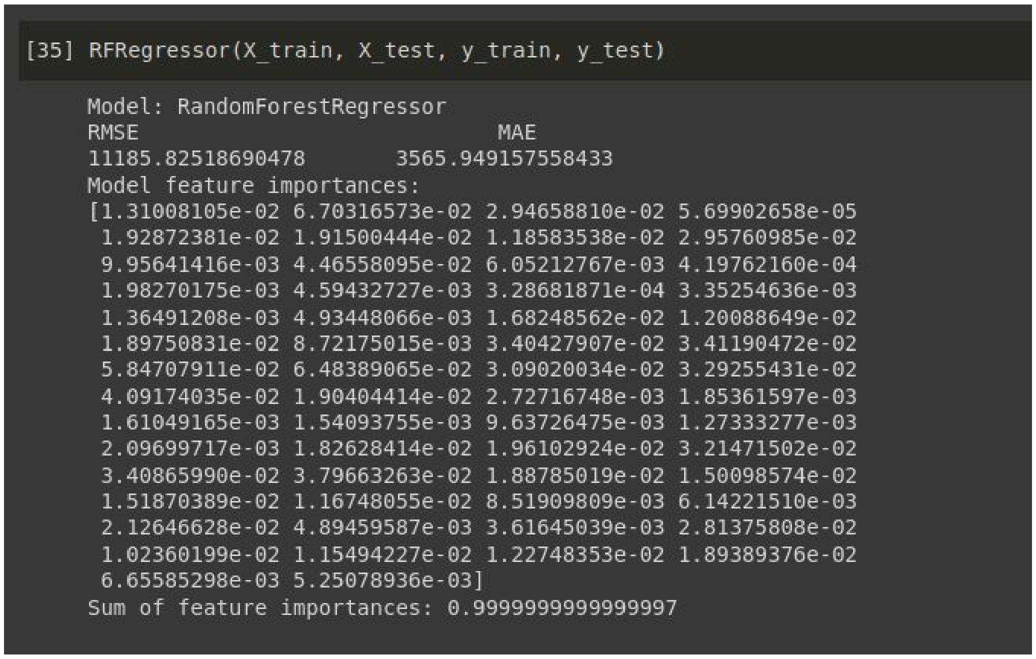
```
[28] test_x_reduced = reduce_dimensions_PCA(test_vectors, X_test)
     test_x_reduced.head()
```

	pca1	pca2
0	-730.963228	-4.888303
1	-2248.262949	-5.771067
2	-1081.224933	-4.046821
3	11263.177934	17.079877
4	-2542.618310	-3.287844



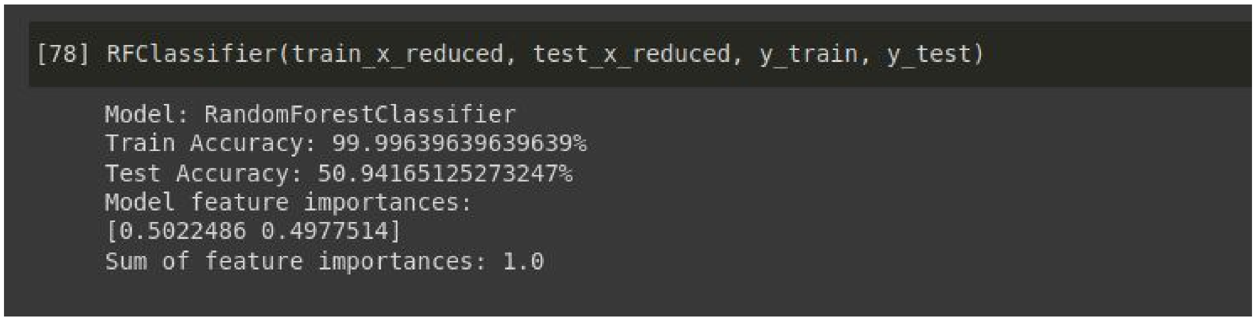
```
RFRegressor(train_x_reduced, test_x_reduced, y_train, y_test)

Model: RandomForestRegressor
RMSE                              MAE
13754.460599496975      4434.709353455523
Model feature importances:
[0.44301843 0.55698157]
Sum of feature importances: 1.0
```



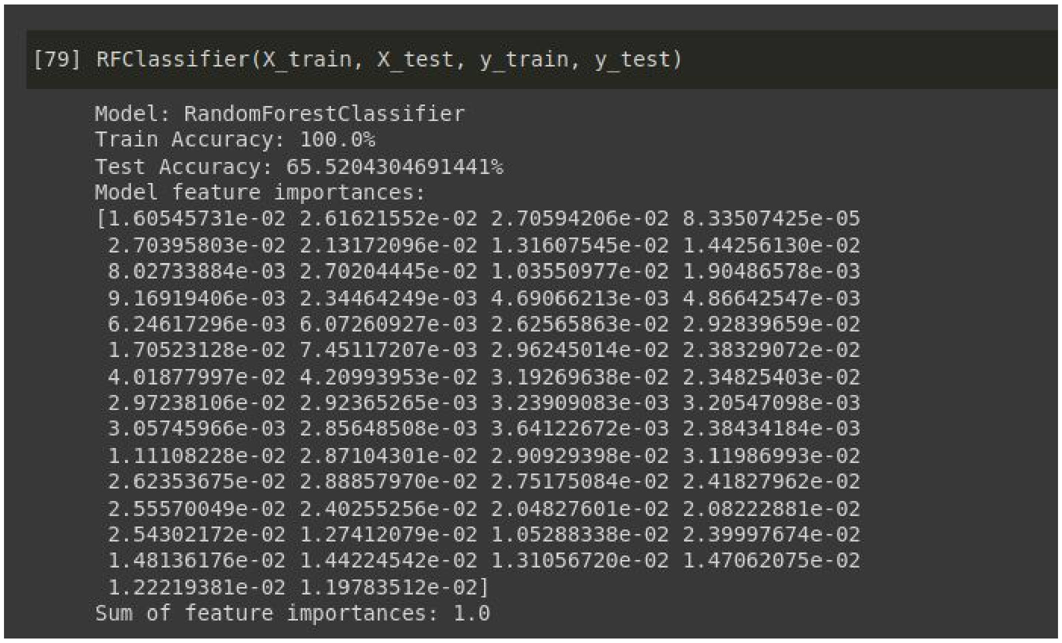
```
[35] RFRegressor(X_train, X_test, y_train, y_test)

     Model: RandomForestRegressor
     RMSE                              MAE
     11185.82518690478       3565.949157558433
     Model feature importances:
     [1.31008105e-02 6.70316573e-02 2.94658810e-02 5.69902658e-05
      1.92872381e-02 1.91500444e-02 1.18583538e-02 2.95760985e-02
      9.95641416e-03 4.46558095e-02 6.05212767e-03 4.19762160e-04
      1.98270175e-03 4.59432727e-03 3.28681871e-04 3.35254636e-03
      1.36491208e-03 4.93448066e-03 1.68248562e-02 1.20088649e-02
      1.89750831e-02 8.72175015e-03 3.40427907e-02 3.41190472e-02
      5.84707911e-02 6.48389065e-02 3.09020034e-02 3.29255431e-02
      4.09174035e-02 1.90404414e-02 2.72716748e-03 1.85361597e-03
      1.61049165e-03 1.54093755e-03 9.63726475e-03 1.27333277e-03
      2.09699717e-03 1.82628414e-02 1.96102924e-02 3.21471502e-02
      3.40865990e-02 3.79663263e-02 1.88785019e-02 1.50098574e-02
      1.51870389e-02 1.16748055e-02 8.51909809e-03 6.14221510e-03
      2.12646628e-02 4.89459587e-03 3.61645039e-03 2.81375808e-02
      1.02360199e-02 1.15494227e-02 1.22748353e-02 1.89389376e-02
      6.65585298e-03 5.25078936e-03]
     Sum of feature importances: 0.9999999999999997
```



```
[78] RFClassifier(train_x_reduced, test_x_reduced, y_train, y_test)

     Model: RandomForestClassifier
     Train Accuracy: 99.99639639639639%
     Test Accuracy: 50.94165125273247%
     Model feature importances:
     [0.5022486 0.4977514]
     Sum of feature importances: 1.0
```



```
[79] RFClassifier(X_train, X_test, y_train, y_test)

     Model: RandomForestClassifier
     Train Accuracy: 100.0%
     Test Accuracy: 65.5204304691441%
     Model feature importances:
     [1.60545731e-02 2.61621552e-02 2.70594206e-02 8.33507425e-05
      2.70395803e-02 2.13172096e-02 1.31607545e-02 1.44256130e-02
      8.02733884e-03 2.70204445e-02 1.03550977e-02 1.90486578e-03
      9.16919406e-03 2.34464249e-03 4.69066213e-03 4.86642547e-03
      6.24617296e-03 6.07260927e-03 2.62565863e-02 2.92839659e-02
      1.70523128e-02 7.45117207e-03 2.96245014e-02 2.38329072e-02
      4.01877997e-02 4.20993953e-02 3.19269638e-02 2.34825403e-02
      2.97238106e-02 2.92365265e-03 3.23909083e-03 3.20547098e-03
      3.05745966e-03 2.85648508e-03 3.64122672e-03 2.38434184e-03
      1.11108228e-02 2.87104301e-02 2.90929398e-02 3.11986993e-02
      2.62353675e-02 2.88857970e-02 2.75175084e-02 2.41827962e-02
      2.55570049e-02 2.40255256e-02 2.04827601e-02 2.08222881e-02
      2.54302172e-02 1.27412079e-02 1.05288338e-02 2.39997674e-02
      1.48136176e-02 1.44224542e-02 1.31056720e-02 1.47062075e-02
      1.22219381e-02 1.19783512e-02]
     Sum of feature importances: 1.0
```

Here, we also compare results with the original dataset. This shows that there quite a bit of difference between the Test Root Mean Squared Error(RMSE) and Mean Absolute Error (MAE) for Regression and the Test Accuracy for Classification of the two.

This can be accounted for by the fact that we only considered the two most important features, and can be resolved by increasing the number of reduced (by PCA) features.

We can see that we achieved results (66%) close to the ones achieved in the original research paper by Fernandez et al. (67%).

Artificial Neural Networks (ANN)

We used ANN's for the binary classification problem to improve our accuracy. We used a single hidden layer of **250 neurons** with **Sigmoid** activation function on both ends using **Adam** as the optimizer at a **learning rate of 0.01** and **epochs ceiled to 1000**. We tested even using **Stochastic Gradient Descent(SGD)** as the optimizer and found that Adam does considerably better than SGD.

We initially tested using frameworks like **Tensorflow** and **Keras** and found the training set accuracy to be almost 100% and the test set accuracy around 60%. But eventually we ended up using **MLPClassifier** of **sklearn** that optimizes the log loss unlike the other two versions where binary cross entropy is optimized.

```
Training Set Accuracy using Tensorflow: 0.9969730377197266
Test Set Accuracy using Tensorflow: 0.5787615083869341
```

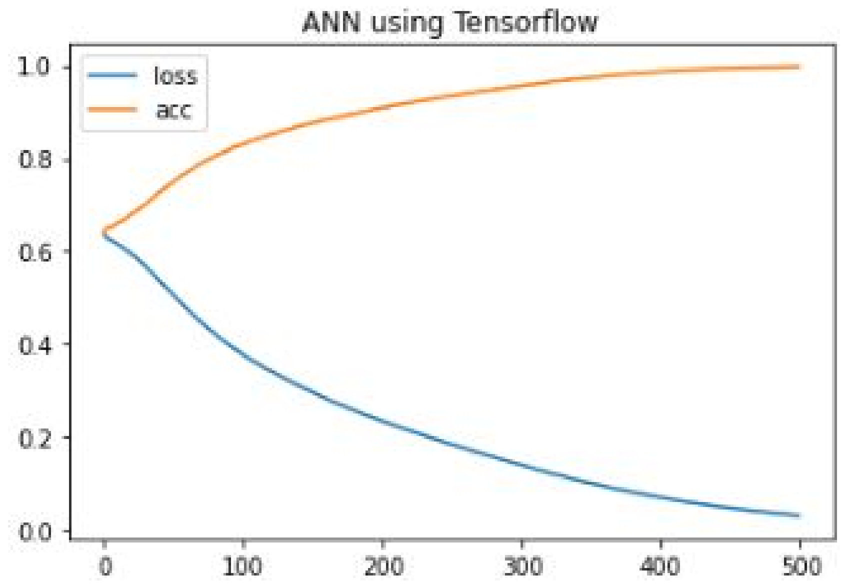
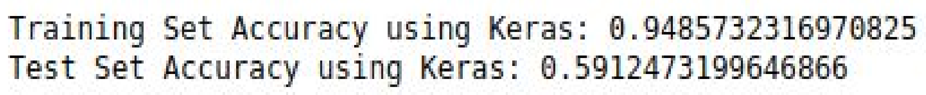


Fig. ANN Implementation using Tensorflow



```
Training Set Accuracy using Keras: 0.9485732316970825
Test Set Accuracy using Keras: 0.5912473199646866
```

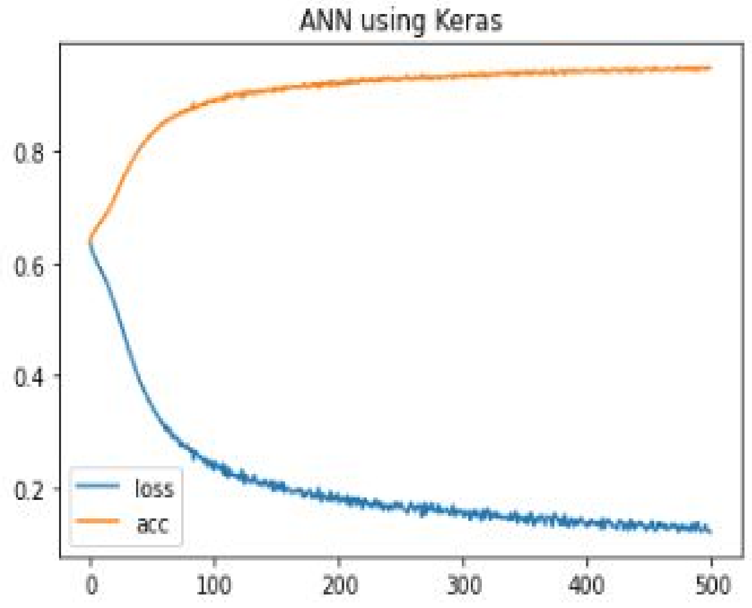
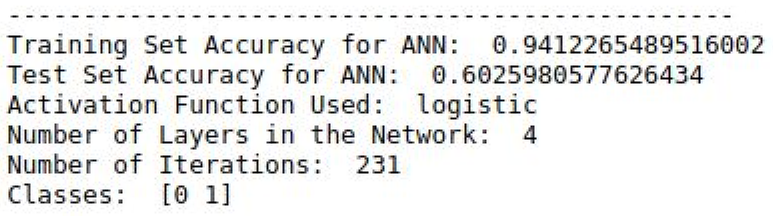


Fig. ANN Implementation using Keras

Eventually we ended up using the **MLPClassifier** of **sklearn** that relies on an underlying Neural Network**.**



```
------------------------------------------------
Training Set Accuracy for ANN:  0.9412265489516002
Test Set Accuracy for ANN:  0.6025980577626434
Activation Function Used:  logistic
Number of Layers in the Network:  4
Number of Iterations:  231
Classes:  [0 1]
```

FIg, model summary

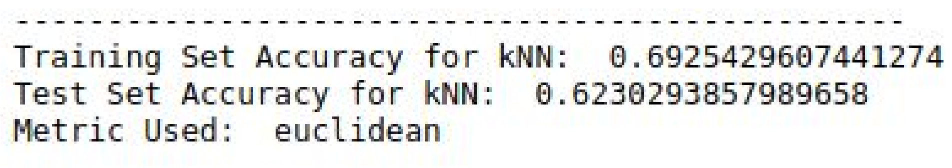
k Nearest Neighbours

Another model we tried out was k Nearest Neighbours, which can be used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. We only used it for the classification problem (shares>1400).

Since this algorithm relies on distance for classification, normalizing the training data can improve its accuracy dramatically. In this method, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

However, since this project was done from a statistical point of view (and not a Machine Learning one), we did not dive deep into this.

On the implementation part, we used **sklearn.neighbors.KNeighborsClassifier** with the distance metric as the **Euclidean Distance** and the **number of neighbours=11**. We initially tested with the number of neighbours=5 and found that n=11 works fairly better.



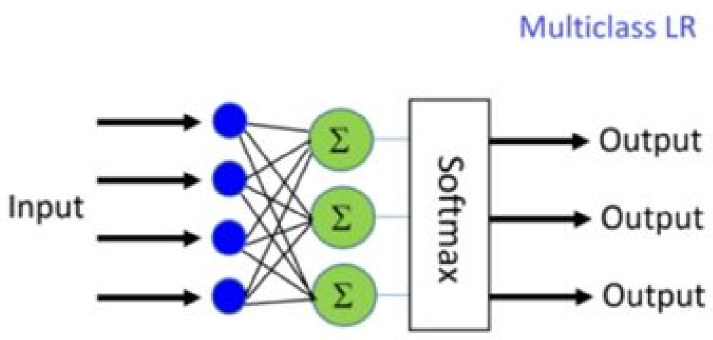
```
------------------------------------------------
Training Set Accuracy for kNN:  0.6925429607441274
Test Set Accuracy for kNN:  0.6230293857989658
Metric Used:  euclidean
```

FIg, model summary

Logistic/SoftMax Regression

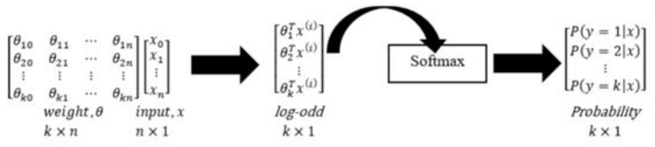
This classifier is an improvement of the MLR classifier. It uses a sigmoid function on top of the linear function as a parametric model. The one done here is softmax regression which is a generalized form of logistic regression classifier.

It can classify multi-class data(class labels are taken as 0,1,2..); , however here we use it for binary classification. For 2 classes it is exactly the same as a logistic regression classifier. It results in piecewise linear discriminants.



The parameters are taken as: (bias term is taken care of by including augmenting input pattern with 1)

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \dots & \theta^{(K)} \\ | & | & | & | \end{bmatrix}$$



Softmax

- $$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

$$\boxed{P(y^{(i)} = k | x^{(i)}; \theta) = \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}}$$

Loss function used is cross-entropy loss function.

Cost function

$$J(\theta) = -\left[\sum_{i=1}^{m} \sum_{k=1}^{K} 1\left\{ y^{(i)} = k \right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} \right]$$

- Each training example contributes to this cost.

Here summation is over the training examples we consider for the update. Here for online update, we consider only 1 training example at a time.

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right]$$

The gradient of cost function is found as above, and the parameter vector is updated.

New_para_vec = Old_para_vec + -learning_rate*grad(cost_fn).

It is run for 5 epochs and optimal parameters are obtained, which can be used to classify any given random vector, we get a probability vector as the output.

The class label corresponding to, maximum posterior probability is outputted by the classifier.

Implementation: All implementation is done from scratch with step by step descriptive functions.

We are able to achieve an accuracy of around 65% on the test set, and 62% on the training set.

Comparing Models

Following is the comparison of the models based on their Root Mean Square Error (RMSE), accuracy and runtime.

Regression Models

Model	RMSE	Mean Absolute Error
Random Forest Regressor	14406.567475720483	4005.5139525791396
Multiple Linear Regression	27515.415159752003	4048.246298215195

Classification Models

Model	Train Accuracy (%)	Test Accuracy (%)
Random Forest Classifier	100	65.5
Neural Network	94.12	60.25
k-Nearest Neighbours	69.25	63.2
Logistic/softmax Regression	62.69	65.33
Multiple Linear Regression	56.6	53.8
Naive Bayes Classifier	52	52

Observations

Out of all the models that we tried, we notice that Random Forests provide the best results, with Train Accuracy of 99.9% and Test Accuracy of 66%.

Further, on careful analysis of the (EDA), the performance of the various models and the feature weights of the best performing models we get the following insights.

In order to increase the popularity of an article:

Increase the:

1. number of embedded links
2. number of images
3. number of videos
4. amount of subjectivity in title
5. amount of subjectivity in content
6. number of words which are more popular
7. references to older articles which have high popularity

Decrease the:

1. number of longer words in the content
2. amount of multi-topic discussion in an article (Articles which talk about multiple topics perform poorly)
3. number of negative words (words with negative polarity)

References

Original Research Paper:

https://www.researchgate.net/publication/283510525_A_Proactive_Intelligent_Decision_Support_System_for_Predicting_the_Popularity_of_Online_News

:Stanford University Research Paper:

http://cs229.stanford.edu/proj2015/328_report.pdf