Homework -4

CSCE 633: Machine Learning

Manav Gurumoorthy

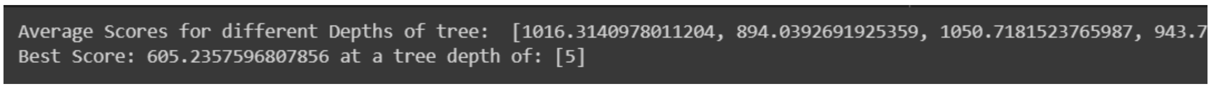
830000011

Introduction:

In this HW, we dealt with Decision Tree Regression and Random Forest Regression. The task was to use a Decision Tree Regression model and Random Forest Regression to predict the number of shares of an article from Mashable.com.
The dataset consists of approximately 40k entries, 60 features to learn from and a final parameter shares that was the result to be computed.

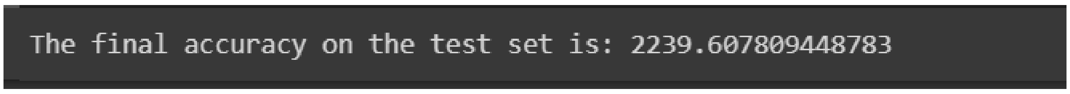
Decision Tree Regression:

In this part, a decision tree regression model was constructed to estimate the number of shares of an article. In order to find a robust model we tuned the depth of the decision tree. This was done using a 5-fold cross validation.
The accuracy of the model was computed using the mean square error metric. The results are as shown below. Depth was varied from 1 to 10.



```
Average Scores for different Depths of tree:  [1016.3140978011204, 894.0392691925359, 1050.7181523765987, 943.7
Best Score: 605.2357596807856 at a tree depth of: [5]
```

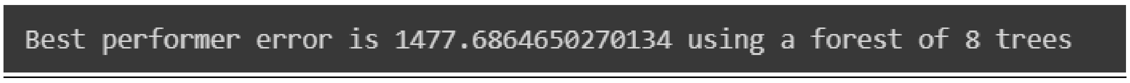
Using this depth of the best performer the model was tested on the test data.



```
The final accuracy on the test set is: 2239.607809448783
```

Random Forest Regression:

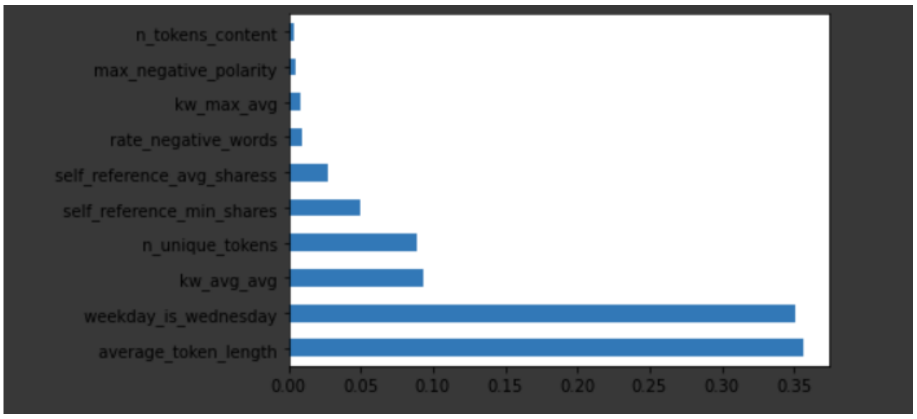
In this section we look to create a random forest for different input features and for a small percentage of the train dataset. We performed 5-fold cross validation on this reduced dataset. The next step was to average the prediction across all tree to find the optimal size of the forest. Once the hyperparameters were determined i.e. the depth of the trees and number of trees in the forest, this forest was used on the test dataset. The final prediction is the average of each tree in the forest.

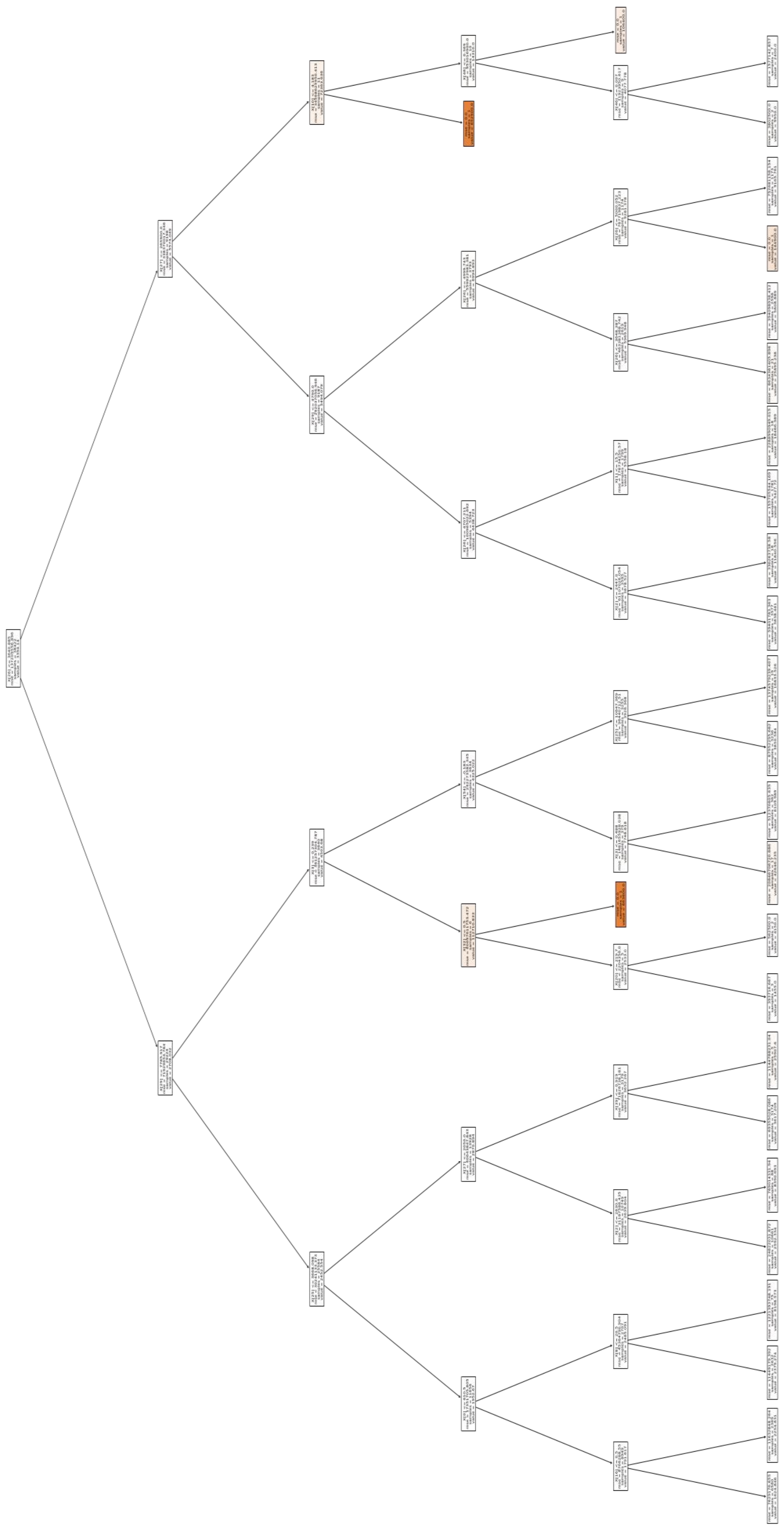


```
Best performer error is 1477.6864650270134 using a forest of 8 trees
```

Feature exploration:

In this part we are interested to determine what the most effective features were in the creation of the model. This was done using feature_importaces_)_ API of the decision tree regression library. This gives us a list of features and their importance. I have plotted the 10 more important features.





The figure above shows the final decision tree. Each parent node represents a feature of higher importance than the child. This tree is 5 layers deep.

APPENDIX A: COLAB WORKBOOK

Importing Data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#IMPORTING THE DATA FROM THE DATASETS.
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt
from skimage import data, io, filters
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

path3 = "/content/drive/My Drive/Homework 4/OnlineNewsPopularityTest.csv"
path = "/content/drive/My Drive/Homework 4/OnlineNewsPopularityTrain.csv"

train_data_df = pd.read_csv(path)
train_data_df = train_data_df.drop('url', axis=1)
train_data = train_data_df.drop(' shares', axis=1)
feature_list = np.array(train_data.columns)
train_outcomes= train_data_df[' shares']

test_data_df = pd.read_csv(path3)
test_data_df = test_data_df.drop('url', axis=1)
test_data = test_data_df.drop(' shares', axis=1)
test_outcomes= test_data_df[' shares']
```

```
X = np.array(train_data)
y = np.array(train_outcomes)
kf = KFold()
kf.get_n_splits(X)
```

```
5
```

```
def mse (y1, y2):
  summation = 0  #variable to store the summation of differences
  n = len(y1) #finding total number of items in list
  for i in range (n):  #looping through each element of the list
    difference = y1[i] - y2[i]  #finding the difference between observed and predicted value
    squared_difference = difference**2  #taking square of the differene
    summation = summation + squared_difference  #taking a sum of all the differences
    MSE = summation/n
    return(MSE)

```

Decision Tree Regression

```
# Fit regression model
k_folds = np.array_split(train_data_df,5)
tree_depth = []
model_score = []
for j in range (1,10):
  mean_square_error= []
  for k in range(5):
    train = k_folds.copy()
    validation = k_folds [k]
    validation_y = validation[' shares']
    validation_y = np.array(validation_y)
    validation = validation.drop(' shares', axis =1)
    validation = np.array(validation)
    del train[k]
    train = pd.concat(train, sort =False)
    train_y = train[' shares']
    train_y = np.array(train_y)
    train = train.drop (' shares',axis=1)
    train = np.array(train)
    regr_1 = DecisionTreeRegressor(criterion="mse",max_depth=j,random_state=0)
    regr_1.fit(train, train_y)
    y_1 = regr_1.predict(validation)
    mean_square_error.append(mse(y_1, validation_y))
  model_score.append(((sum(mean_square_error))/(len(mean_square_error))))
i=(np.argmin(model_score))
print("Average Scores for different Depths of tree: ", model_score)
print("Best Score: {0} at a tree depth of: {1}".format(model_score[i],[i]))
```

```
Average Scores for different Depths of tree:  [1016.3140978011204, 894.0392691925359, 1050.7181523765987, 943.7684687466854, 766
Best Score: 605.2357596807856 at a tree depth of: [5]
```

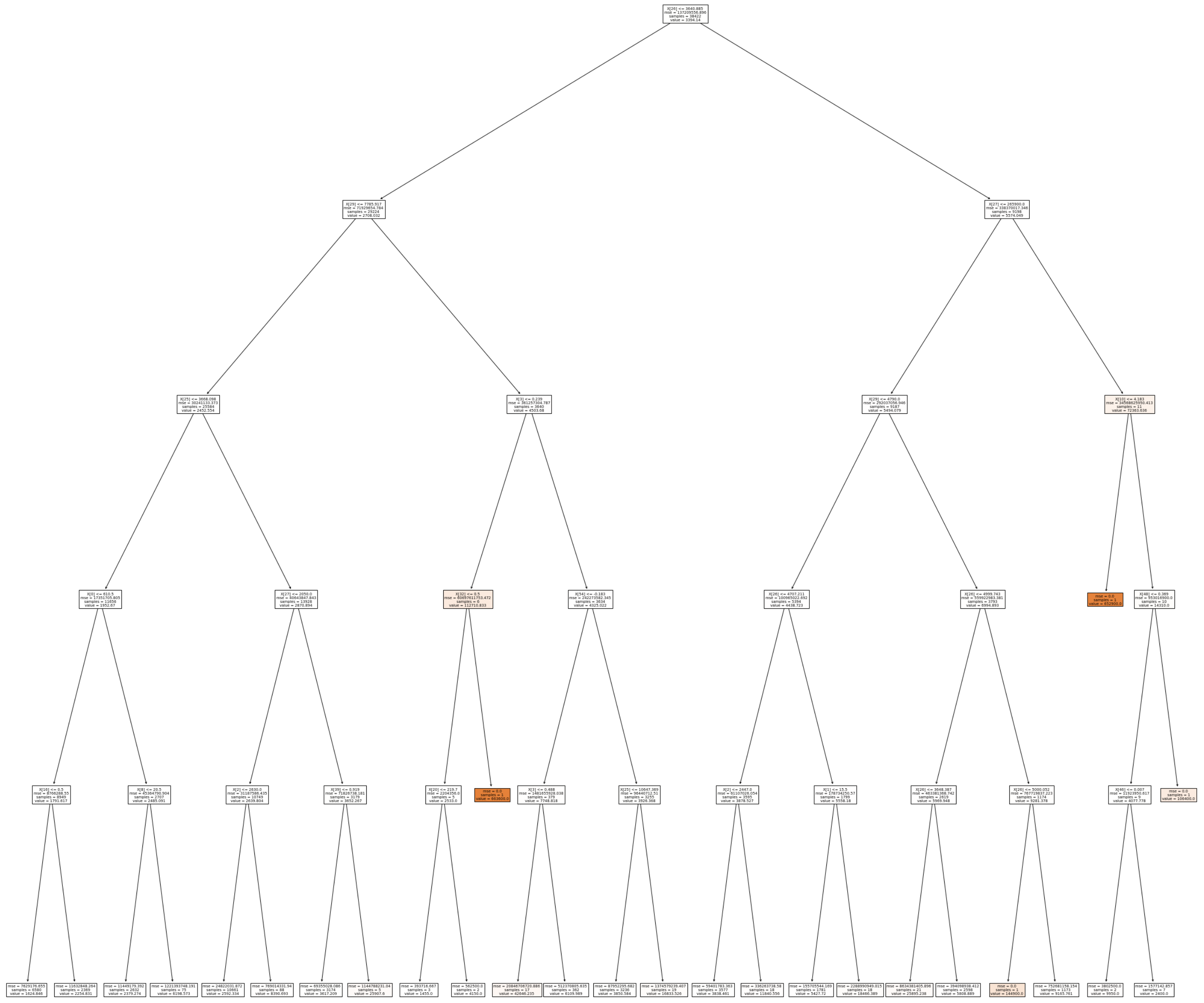
```
print(np.argmin(model_score))
```

```
5
```

```
regr_2 = DecisionTreeRegressor(criterion="mse",max_depth=i,random_state=0)
regr_2.fit(X,y)
y_2 = regr_2.predict(test_data)
mean_square_error=(mse(y_2,test_outcomes))
print(mean_square_error)
```

```
2611.1073794518447
```

```
#PLOTTING THIS JUST FOR FUN.
from sklearn import tree
plt.figure(figsize=(40,40))
#tree.plot_tree(regr_2.fit(test_data, test_outcomes))
tree.plot_tree(regr_2, filled=True)
plt.show()
```


X[26] <= 3640.885
mse = 137209556.896
samples = 38422
value = 3394.14
X[29] <= 7785.917
mse = 71929654.784
samples = 29224
value = 2708.032
X[27] <= 265900.0
mse = 338370017.346
samples = 9198
value = 5574.049
X[25] <= 3668.098
mse = 30241133.373
samples = 25584
value = 2452.554
X[3] <= 0.239
mse = 361257304.787
samples = 3640
value = 4503.68
X[29] <= 4790.0
mse = 292037056.946
samples = 9187
value = 5494.079
X[10] <= 4.183
mse = 34568625950.413
samples = 11
value = 72363.636
X[0] <= 610.5
mse = 17351705.805
samples = 11656
value = 1952.67
X[27] <= 2050.0
mse = 40643847.843
samples = 13928
value = 2870.894
X[32] <= 0.5
mse = 60697611753.472
samples = 6
value = 112710.833
X[54] <= -0.183
mse = 242273582.345
samples = 3634
value = 4325.022
X[26] <= 4707.211
mse = 100965022.692
samples = 5394
value = 4438.723
X[26] <= 4999.743
mse = 559922983.381
samples = 3793
value = 6994.893
mse = 0.0
samples = 1
value = 652900.0
X[48] <= 0.369
mse = 953016900.0
samples = 10
value = 14310.0
X[16] <= 0.5
mse = 8766288.55
samples = 8949
value = 1791.617
X[8] <= 20.5
mse = 45364790.904
samples = 2707
value = 2485.091
X[2] <= 2630.0
mse = 31187586.435
samples = 10749
value = 2639.804
X[39] <= 0.919
mse = 71826738.181
samples = 3179
value = 3652.267
X[20] <= 219.7
mse = 2204356.0
samples = 5
value = 2533.0
mse = 0.0
samples = 1
value = 663600.0
X[3] <= 0.488
mse = 1481655928.038
samples = 379
value = 7748.818
X[25] <= 10647.369
mse = 96440712.51
samples = 3255
value = 3926.368
X[2] <= 2447.0
mse = 61107026.054
samples = 3595
value = 3878.527
X[1] <= 15.5
mse = 178734250.57
samples = 1799
value = 5558.18
X[26] <= 3648.387
mse = 463381368.742
samples = 2619
value = 5969.948
X[26] <= 5000.052
mse = 767719837.223
samples = 1174
value = 9281.378
X[46] <= 0.007
mse = 11923950.617
samples = 9
value = 4077.778
mse = 0.0
samples = 1
value = 106400.0
mse = 7629176.655
samples = 6580
value = 1624.846
mse = 11632848.264
samples = 2369
value = 2254.831
mse = 11449179.392
samples = 2632
value = 2379.274
mse = 1221393748.191
samples = 75
value = 6198.573
mse = 24822031.872
samples = 10661
value = 2592.334
mse = 769014331.94
samples = 88
value = 8390.693
mse = 69355028.086
samples = 3174
value = 3617.209
mse = 1144788231.04
samples = 5
value = 25907.6
mse = 393716.667
samples = 3
value = 1455.0
mse = 562500.0
samples = 2
value = 4150.0
mse = 20846706720.886
samples = 17
value = 42646.235
mse = 512370805.635
samples = 362
value = 6109.989
mse = 87952295.682
samples = 3236
value = 3850.584
mse = 1374579239.407
samples = 19
value = 16833.526
mse = 59401783.363
samples = 3577
value = 3838.461
mse = 336263738.58
samples = 18
value = 11840.556
mse = 155705544.169
samples = 1781
value = 5427.72
mse = 2288990949.015
samples = 18
value = 18466.389
mse = 8634381405.896
samples = 21
value = 25895.238
mse = 394098938.412
samples = 2598
value = 5808.889
mse = 0.0
samples = 1
value = 144900.0
mse = 752681158.154
samples = 1173
value = 9165.761
mse = 3802500.0
samples = 2
value = 9950.0
mse = 1577142.857
samples = 7
value = 2400.0

Random Forest Regression

```python
# Fit regression model
def tree_regressor ():
  data_in = train_data_df.sample(frac=0.3,axis=0)
  k_folds = np.array_split(data_in,5)
  y = data_in[' shares']
  y = np.array(y)
  X = data_in.drop(' shares', axis=1)
  X = np.array(X)
  tree_depth = []
  model_score = []
  y_output = []
  for j in range (1,11):
    mean_square_error= []
    predicted_output = []
    for k in range(5):
      train = k_folds.copy()
      validation = k_folds [k]
      validation_y = validation[' shares']
      validation_y = np.array(validation_y)
      validation = validation.drop(' shares', axis =1)
      validation = np.array(validation)
      del train[k]
      train = pd.concat(train, sort=False)
      train_y = train[' shares']
      train_y = np.array(train_y)
      train = train.drop (' shares',axis=1)
      train = np.array(train)
      regr_1 = DecisionTreeRegressor(criterion="mse",max_depth=j,max_features="log2")
      regr_1.fit(train, train_y)
      y_1 = regr_1.predict(validation)
      mean_square_error.append(mse(y_1, validation_y))
    model_score.append(((sum(mean_square_error))/(len(mean_square_error))))
  i=(np.argmin(model_score))
  regr_out = DecisionTreeRegressor(criterion="mse",max_depth=j,max_features="log2")
  regr_out.fit(X,y)
  y_out = regr_out.predict(test_data)
  return(y_out)
```

```
def random_forest (no_of_trees):
  y_outcomes = []
  for k in range (1, no_of_trees):
    y_outcomes.append(tree_regressor())
  summed_outcomes = np.zeros(len(y_outcomes[0]))
  for i in range(len(y_outcomes)):
    for j in range(len(y_outcomes[i])):
      summed_outcomes[j] = summed_outcomes[j] + y_outcomes[i][j]
  summed_outcomes = summed_outcomes/(len(y_outcomes))
  result = (mse(summed_outcomes,test_outcomes))
  return(result)
```

```
best_forest = []
for i in range (2,10):
```

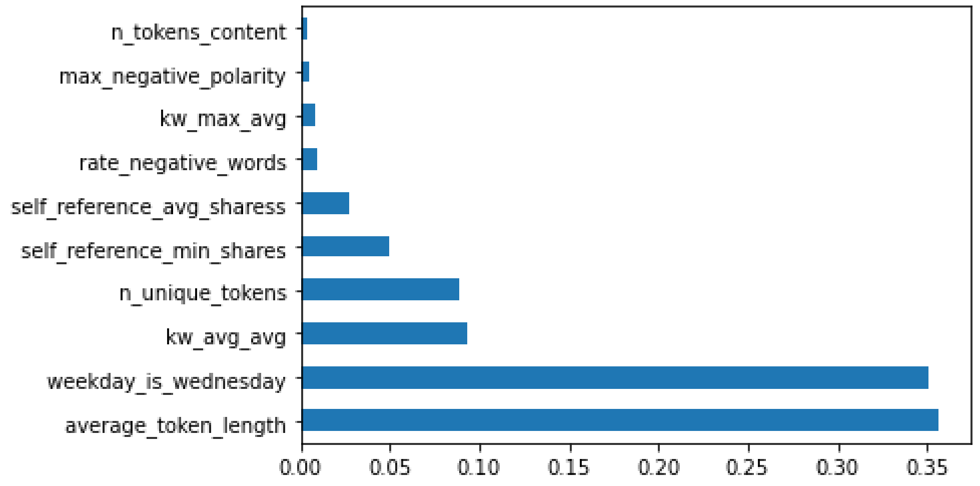
```
for i in range (2,10):
  temp = (random_forest(i))
  best_forest.append(temp)
i=np.argmin(best_forest)
print("Best performer error is {0} using a forest of {1} trees".format(best_forest[i],i+1))
```

```
Best performer error is 1477.6864650270134 using a forest of 8 trees
```

Feature Exploration

```
print(regr_2.feature_importances_)
#plot graph of feature importances for better visualization
feat_importances = pd.Series(regr_2.feature_importances_,index=feature_list)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[9.62466339e-04 2.91732094e-03 3.93012428e-03 8.86356188e-02
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 1.02432754e-03 0.00000000e+00 3.56996536e-01 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 6.65716402e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
 8.39288447e-06 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 8.21802410e-03 9.29751009e-02 4.97306617e-02
 0.00000000e+00 2.69224268e-02 0.00000000e+00 0.00000000e+00
 3.50689021e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 2.38854951e-03
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 8.53870264e-05 0.00000000e+00
 9.07391896e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 4.77640717e-03 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00]
```


n_tokens_content
max_negative_polarity
kw_max_avg
rate_negative_words
self_reference_avg_sharess
self_reference_min_shares
n_unique_tokens
kw_avg_avg
weekday_is_wednesday
average_token_length