Predicting Virality with Extreme Gradient Boosting on Online News Popularity Data

Ednalyn C. De Dios

D214

September 11, 2023

Western Governors University

Predicting Virality with Extreme Gradient Boosting on Online News Popularity Data

In this digital age that we’re in, understanding the different factors that contribute to the popularity of online news articles is a crucial endeavor for media organizations, marketing professionals, and content creators alike. Boundless amount of data exists and presents a challenge of distilling patterns to uncover hidden insights regarding user behavior. Furthermore, identifying key factors that determine an online news article’s popularity has become the holy grail of many, including data scientists. Machine learning techniques like XGBoost help uncover these hidden insights and translate them into actionable nuggets of information that stakeholders can act upon.

XGBoost, which stands for eXtreme Gradient Boosting, is a powerful and versatile machine learning algorithm that has gained popularity in recent years due to its effectiveness in handling a wide range of predictive tasks. Created by Tianqi Chen, XGBoost is “a scalable tree boosting system” (2023) equipped with both sequential and parallel architectures (Suginoo, 2022). A supervised learning algorithm, gradient boosting predicts “a target variable by combining the estimates of a set of simpler, weaker models” (How XGBoost Works, n.d.).

# A – Research Question

In this study, eXtreme Gradient Boosting or XGBoost is used to analyze the Online News Popularity Data by Fernandes et al. (2015) and predict the popularity of online news articles. The aim is to construct a model with more than 65% accuracy and an AUC score of above 60%. The secondary goal of this study is to identify which attributes in the available data are key factors driving the number of social media shares. Stakeholders would like to know this information to optimize their content for “going viral.” This study will examine the different relationships between different attributes of the data in relation to the designation of whether the online news article is popular or not, as defined by the number of times the article has been shared on social media channels. Thus, the question can then be summarized as follows: Can gradient boosting be constructed based solely on the research data?

The null hypothesis of the research question is that gradient boosting cannot be made from the Online News Popularity dataset. For example, an optimized gradient boosting model using XGBoost fails to achieve an accuracy score of more than 65% nor an AUC score of more than 60%. The alternative hypothesis is that an optimized gradient boosting model can be made from the Online News Popularity dataset. For example, an optimized gradient boosting model using XGBoost achieves an accuracy score of more than 65% with an AUC score of more than 60%.

# B – Data Collection

This study uses the “Online News Popularity” dataset which is publicly available from the UC Irvine Machine Learning Repository project (Fernandes et al., 2015). The dataset contains statistics on articles published by Mashable.com. The dataset contains 39,797 records and 61 attributes, of which 58 are predictive, two are non-predictive, and one goal field. The dataset is publicly available to the public and licensed under [Creative Commons Attribution 4.0 International](https://creativecommons.org/licenses/by/4.0/legalcode) (CC BY 4.0). The following table lists the name, type, and description of the attributes that were included in this study.

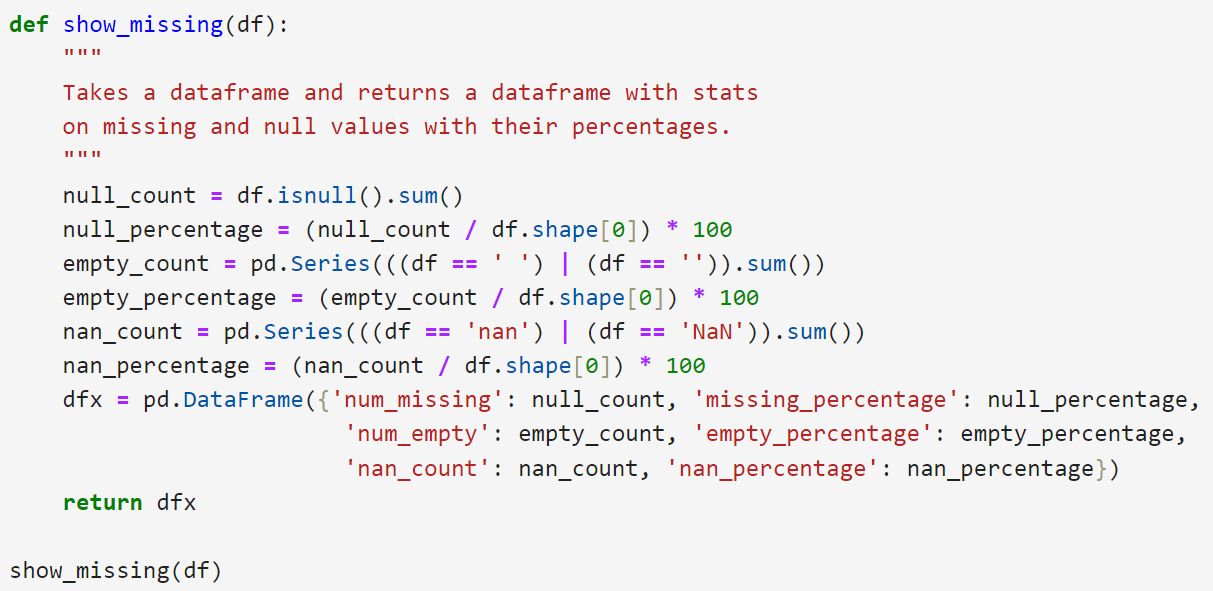
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| url | Categorical | URL of the article |
| timedelta | Continuous | Days between the article publication and the dataset acquisition |
| n\_tokens\_title | Continuous | Number of words in the title |
| n\_tokens\_content | Continuous | Number of words in the content |
| n\_unique\_tokens | Continuous | Rate of unique words in the content |
| n\_non\_stop\_words | Continuous | Rate of non-stop words in the content |
| n\_non\_stop\_unique\_tokens | Continuous | Rate of unique non-stop words in the  content |
| num\_hrefs | Continuous | Number of links |
| num\_self\_hrefs | Continuous | Number of links to other articles published by Mashable |
| num\_imgs | Continuous | Number of images |
| num\_videos | Continuous | Number of videos |
| average\_token\_length | Continuous | Average length of the words in the content |
| num\_keywords | Continuous | Number of keywords in the metadata |
| data\_channel\_is\_lifestyle | Categorical | Is data channel 'Lifestyle'? |
| data\_channel\_is\_entertainment | Categorical | Is data channel 'Entertainment'? |
| data\_channel\_is\_bus | Categorical | Is data channel 'Business'? |
| data\_channel\_is\_socmed | Categorical | Is data channel 'Social Media'? |
| data\_channel\_is\_tech | Categorical | Is data channel 'Tech'? |
| data\_channel\_is\_world | Categorical | Is data channel 'World'? |
| kw\_min\_min | Continuous | Worst keyword (min. shares) |
| kw\_max\_min | Continuous | Worst keyword (max. shares) |
| kw\_avg\_min | Continuous | Worst keyword (avg. shares) |
| kw\_min\_max | Continuous | Best keyword (min. shares) |
| kw\_max\_max | Continuous | Best keyword (max. shares) |
| kw\_avg\_max | Continuous | Best keyword (avg. shares) |
| kw\_min\_avg | Continuous | Avg. keyword (min. shares) |
| kw\_max\_avg | Continuous | Avg. keyword (max. shares) |
| kw\_avg\_avg | Continuous | Avg. keyword (avg. shares) |
| self\_reference\_min\_shares | Continuous | Min. shares of referenced articles in Mashable |
| self\_reference\_max\_shares | Continuous | Max. shares of referenced articles in Mashable |
| self\_reference\_avg\_sharess | Continuous | Avg. shares of referenced articles in Mashable |
| weekday\_is\_monday | Categorical | Was the article published on a Monday? |
| weekday\_is\_tuesday | Categorical | Was the article published on a Tuesday? |
| weekday\_is\_wednesday | Categorical | Was the article published on a Wednesday? |
| weekday\_is\_thursday | Categorical | Was the article published on a Thursday? |
| weekday\_is\_friday | Categorical | Was the article published on a Friday? |
| weekday\_is\_saturday | Categorical | Was the article published on a Saturday? |
| weekday\_is\_sunday | Categorical | Was the article published on a Sunday? |
| is\_weekend | Categorical | Was the article published on the weekend? |
| LDA\_00 | Categorical | Closeness to LDA topic 0 |
| LDA\_01 | Categorical | Closeness to LDA topic 1 |
| LDA\_02 | Categorical | Closeness to LDA topic 2 |
| LDA\_03 | Categorical | Closeness to LDA topic 3 |
| LDA\_04 | Categorical | Closeness to LDA topic 4 |
| global\_subjectivity | Continuous | Text subjectivity |
| global\_sentiment\_polarity | Continuous | Text sentiment polarity |
| global\_rate\_positive\_words | Continuous | Rate of positive words in the content |
| global\_rate\_negative\_words | Continuous | Rate of negative words in the content |
| rate\_positive\_words | Continuous | Rate of positive words among non-neutral tokens |
| rate\_negative\_words | Continuous | Rate of negative words among non-neutral tokens |
| avg\_positive\_polarity | Continuous | Avg. polarity of positive words |
| min\_positive\_polarity | Continuous | Min. polarity of positive words |
| max\_positive\_polarity | Continuous | Max. polarity of positive words |
| avg\_negative\_polarity | Continuous | Avg. polarity of negative words |
| min\_negative\_polarity | Continuous | Min. polarity of negative words |
| max\_negative\_polarity | Continuous | Max. polarity of negative words |
| title\_subjectivity | Continuous | Title subjectivity |
| title\_sentiment\_polarity | Continuous | Title polarity |
| abs\_title\_subjectivity | Continuous | Absolute subjectivity level |
| abs\_title\_sentiment\_polarity | Continuous | Absolute polarity level |
| shares | Continuous | Number of shares (target) |

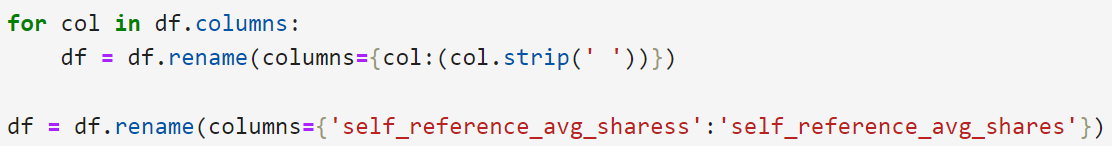
There are several advantages with the chosen dataset. There are no missing values, the class distribution is relatively balanced, and the categorical variables have already been encoded into numerical form. However, one observed disadvantage is the sheer size of the dataset. One trial run of the experiment took almost three hours to run hyperparameter tuning. The challenge of long running time was overcome by setting the tree\_method to ‘gpu\_hist’ and limiting the range of the parameter search space.



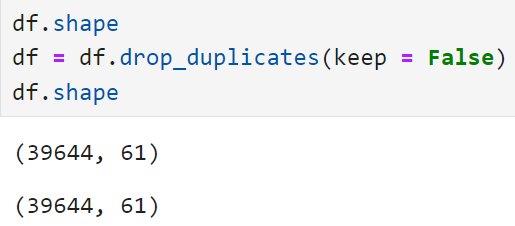
C – Data Extraction and Preparation

The next phase of the analysis is data preparation. The tool used is Python 3.9.9 and Jupyter Notebook 7.0.2 was used as the interactive development environment. Python was chosen for its versatility and easy-to-learn syntax. Jupyter Notebook was chosen for its markdown capability. One disadvantage of Python is performance while Jupyter makes good code versioning very difficult (Mueller, 2018).

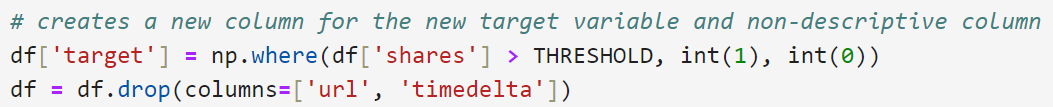
After reading the data into a dataframe, the next step is to check for any missing values.

Next is correcting the column names.

Then, duplicates are dropped if any. There weren’t any as shown by df.shape before and after the operation.



The target variable was created by applying a condition on the ‘shares’ variable and designating 1 or 0 depending on the THRESHOLD value. The threshold value for this notebook is 1400.



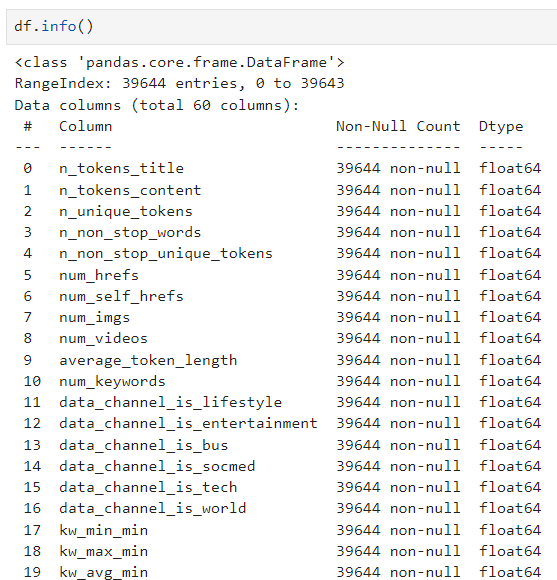
Finally, the cleaned and prepared dataset is exported using pandas’ to\_csv() method.P278#y1

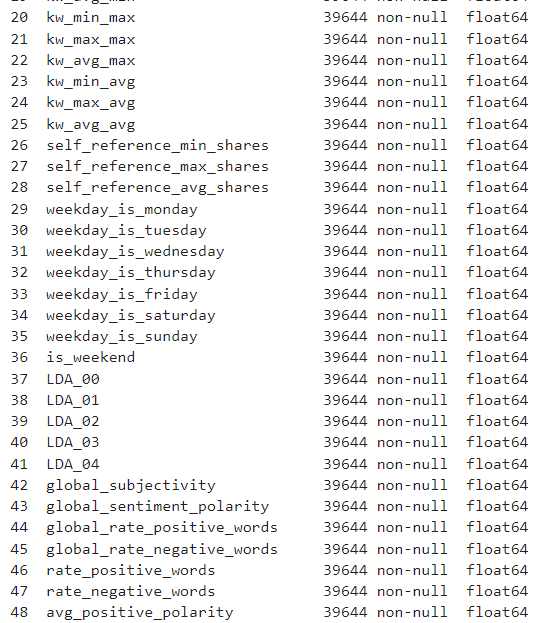
D – Analysis

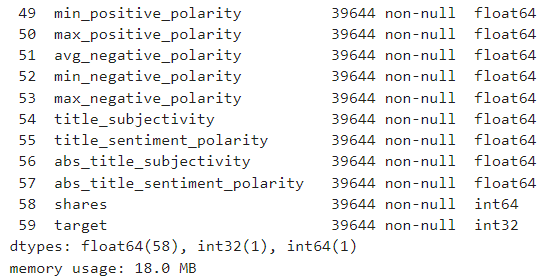
Once the dataset is prepared, the next step in the analysis is to conduct EDA or explanatory data analysis. The high-levels steps are:

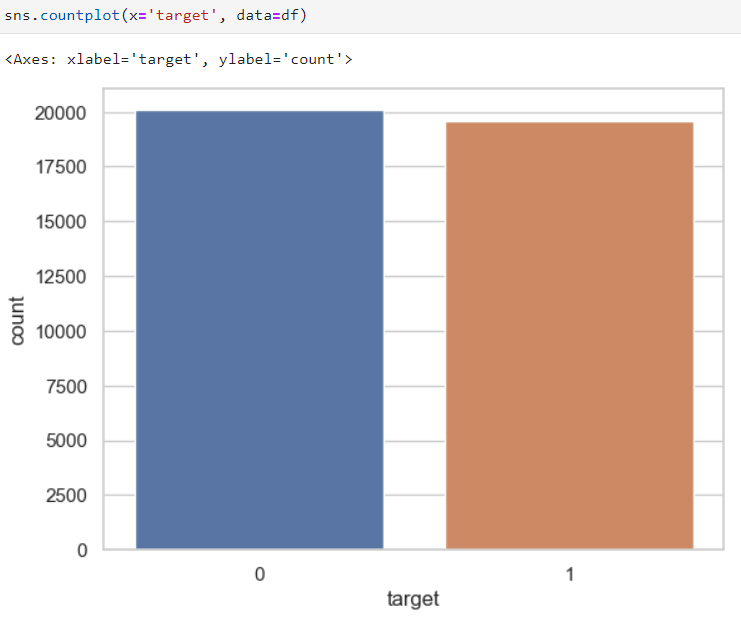
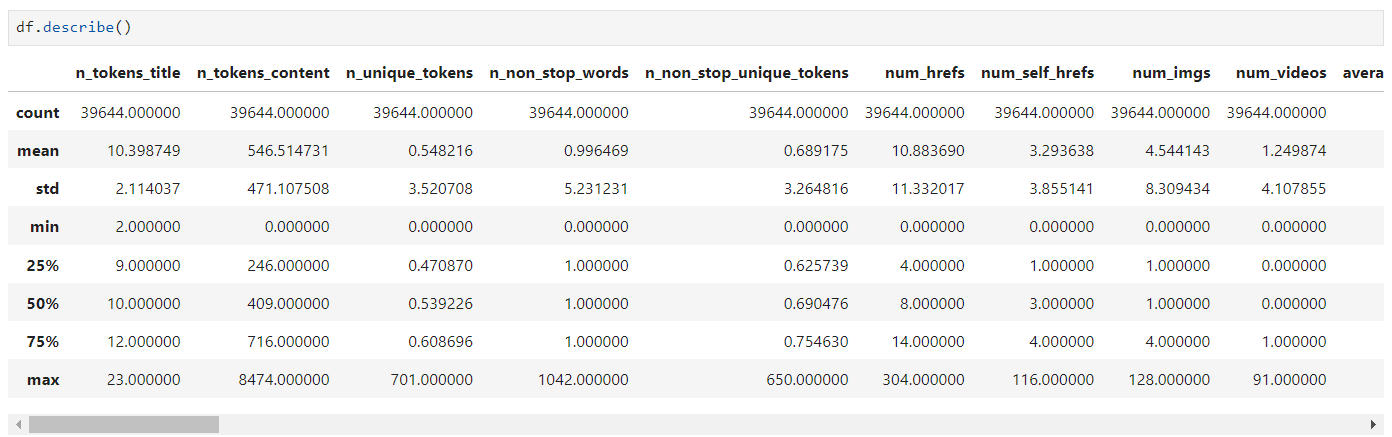
1. Get familiar with the data
2. Review class distribution
3. Get summary statistics
4. Remove outliers
5. Compare the interactions of the variables

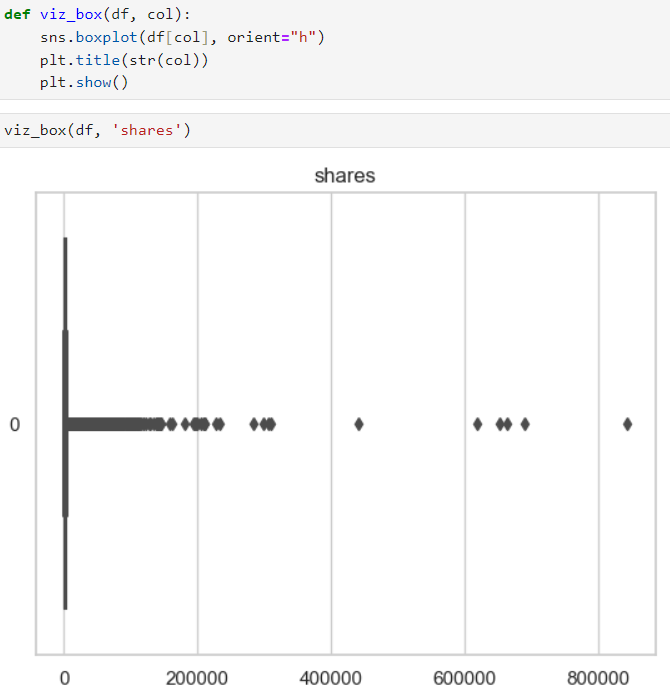


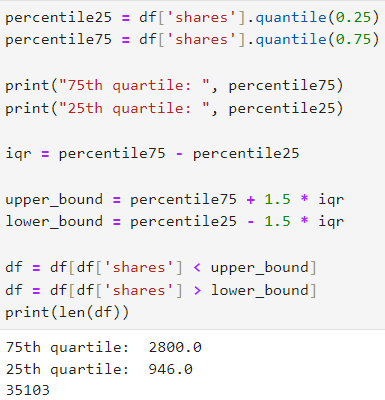
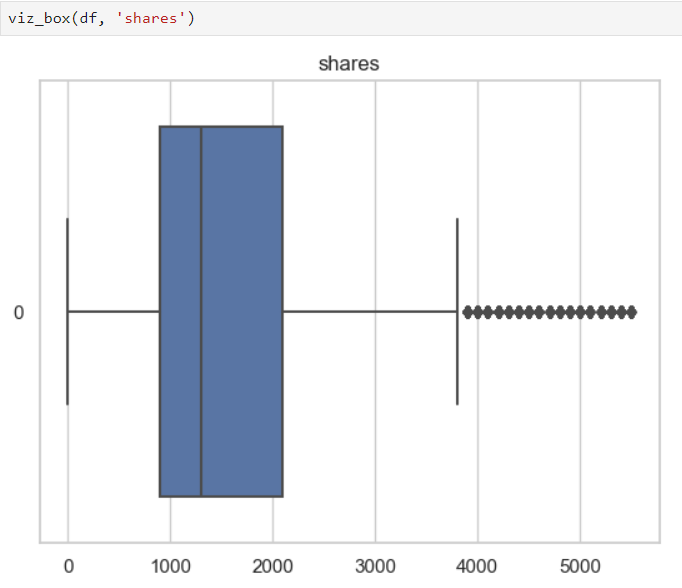


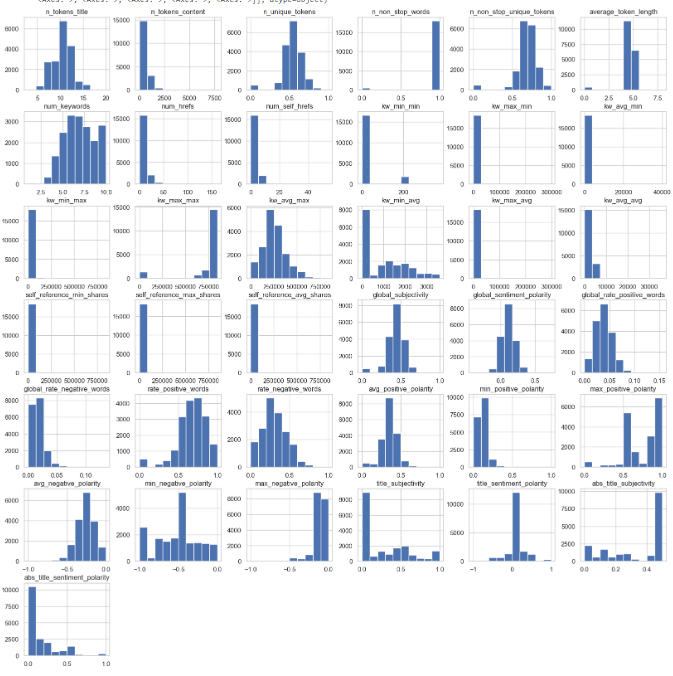


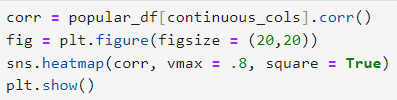


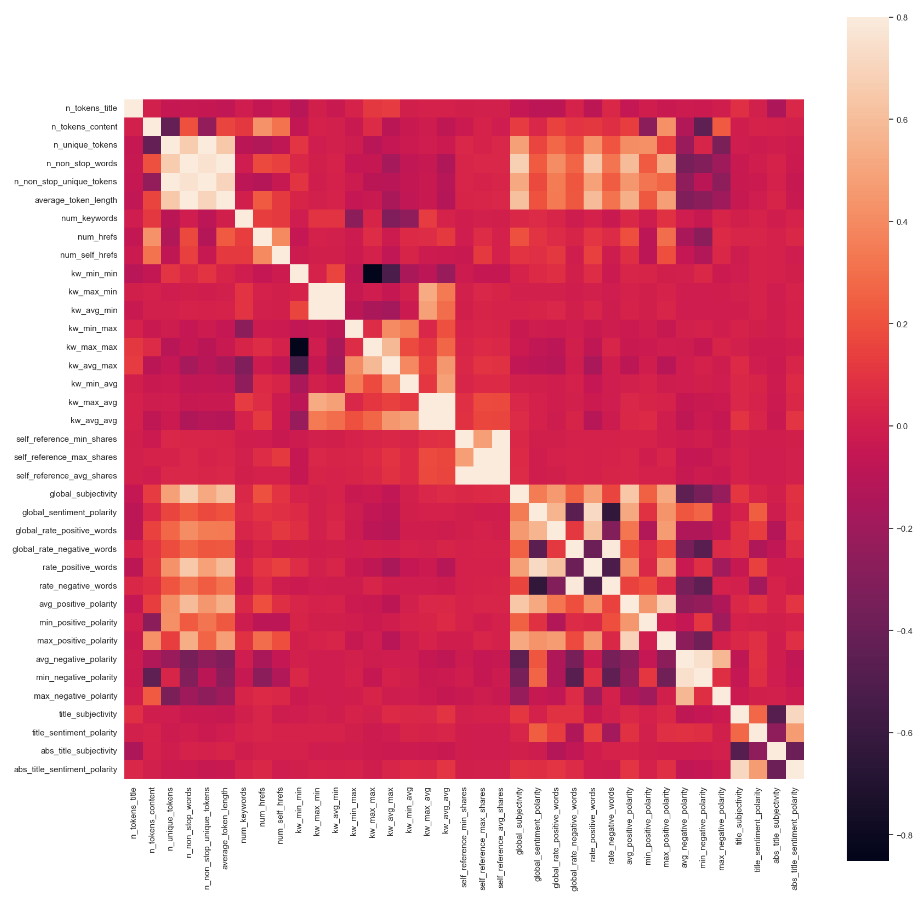
As shown in the graph below, the dataset has a relatively balanced class.

 Next is to conduct a visual inspection of the boxplot to look for outliers in the dataset.

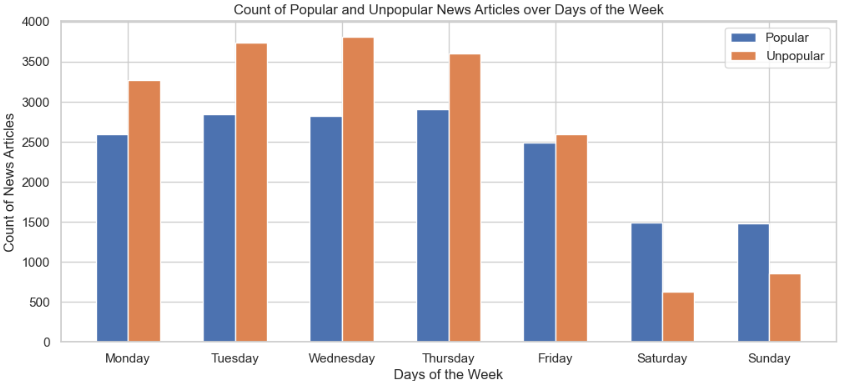
 After confirming the existence of outliers visually, the interquartile range or IQR was calculated so that the records outside the lower and upper bound could be removed.

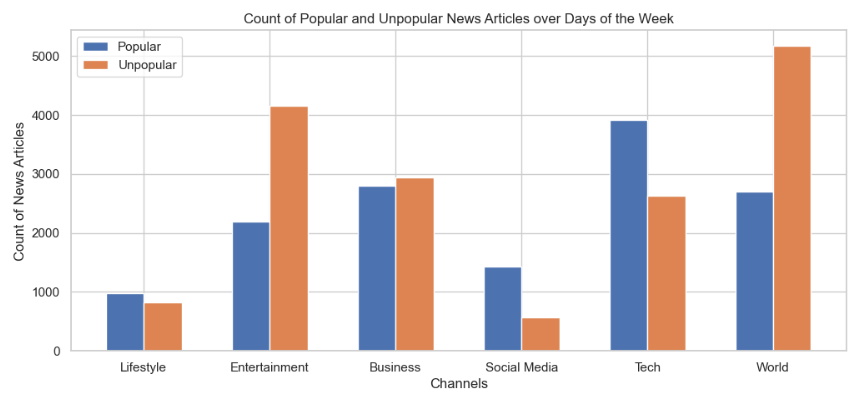
 Next is the plotting of histograms for the continuous variables. As shown, the continuous variables are not distributed normally.

 The correlation heatmap shows a few interesting relationships between several variables.

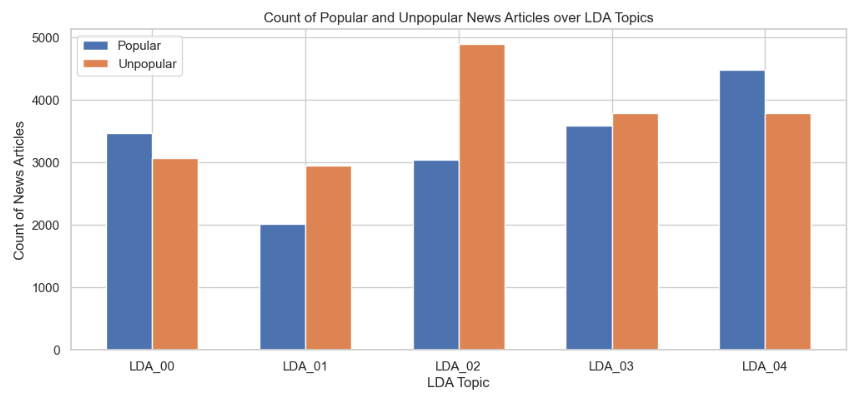


According to the graph below, the weekend is a slow news cycle for both popular and unpopular articles. It is worth noting that articles published during the weekend are more likely to be popular those published during the week.

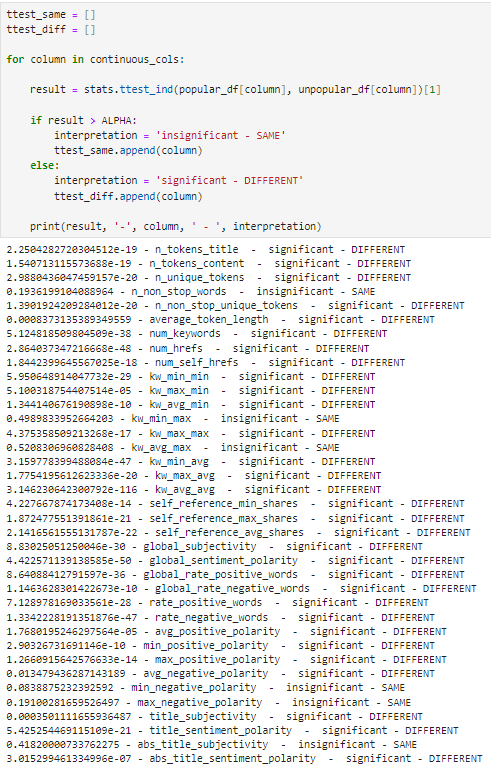


 Evident in the graph below, the topics of Tech, Business, and Entertainment dominate both popular and unpopular articles. Volume wise, the difference between the count of popular and unpopular articles in the World and Entertainment is noteworthy.

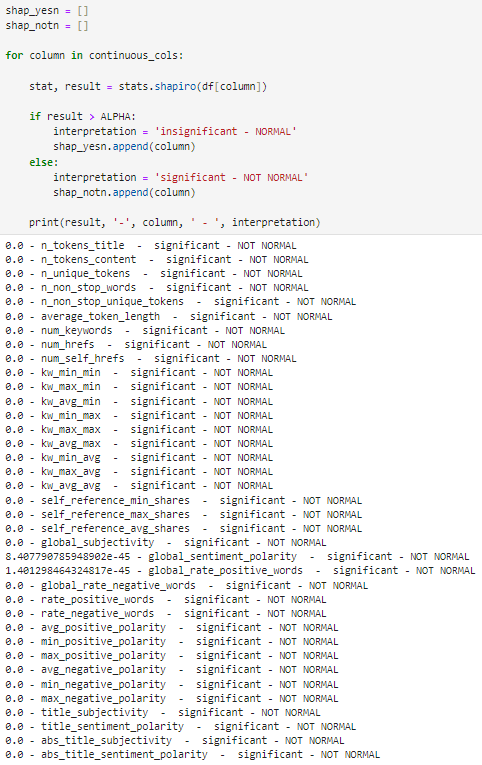
Similarly, LDA Topic #2 shows the same imbalance between popular and unpopular articles.



Conducting a T-test revealed more significant differences between the popular and unpopular groups than insignificant ones. In this case, n\_non\_stop\_words, kw\_min\_max, kw\_avg\_max, min\_negative\_polarity, max\_negative polarity, and abs\_title\_subjectivity have insignificant differences in samples.



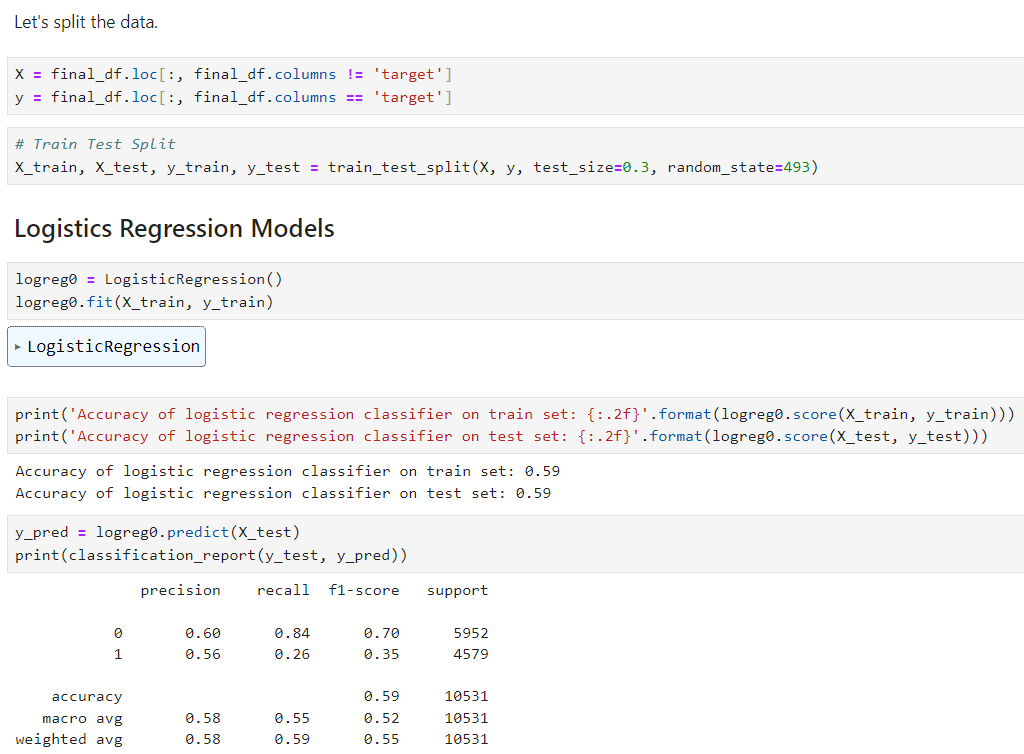
Confirming the visual inspection earlier, Shapiro tests indicate that none of the variables are distributed normally.



P343#y1

The exploration of the dataset involved both visual exploration and statistical testing. An advantage of visualization is the ease it provides the reader to grasp the characteristics of the data that is being inspected. In addition, the output of the statistical testing eliminates guesswork by supplying a statistic and p-value. One disadvantage is that visual inspection can only go so far. It does not provide a value up front without extensive coding of matplotlib methods.

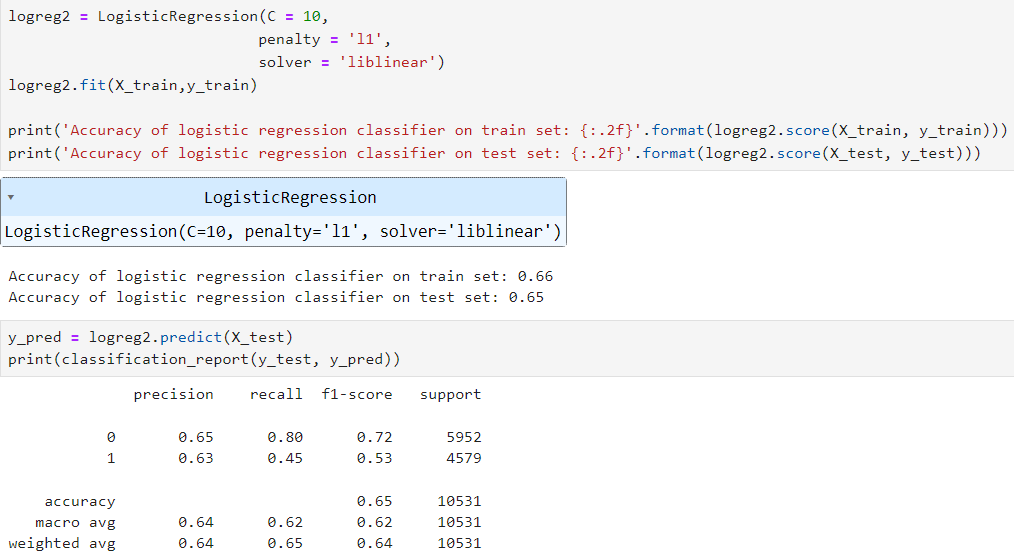
These are the steps involved in the modeling part of the analysis:

1. Splitting the dataset into training and test sets
2. Building logistic regression models for reference
3. Building XGBoost classifier models
4. Extracting feature importance based on the best XGBoost model

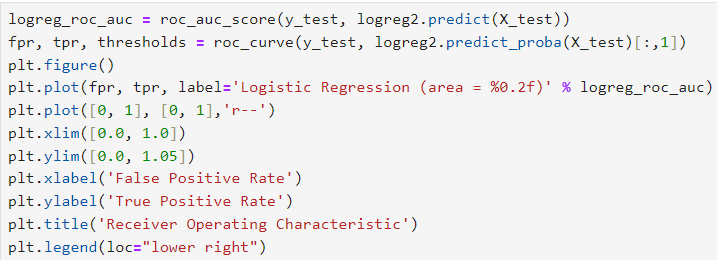
After splitting the data and building the initial logistic regression model, GridSearchCV was utilized to determine the ideal parameters that maximize the accuracy of the logistic regression model.



In the following code, the final logistic regression model is built using the best parameters given by grid search. The accuracy of the final logistic regression model is 0.65, a value that is relatively close to 0.64 which is the accuracy of the training set. This means that the model generalizes well.



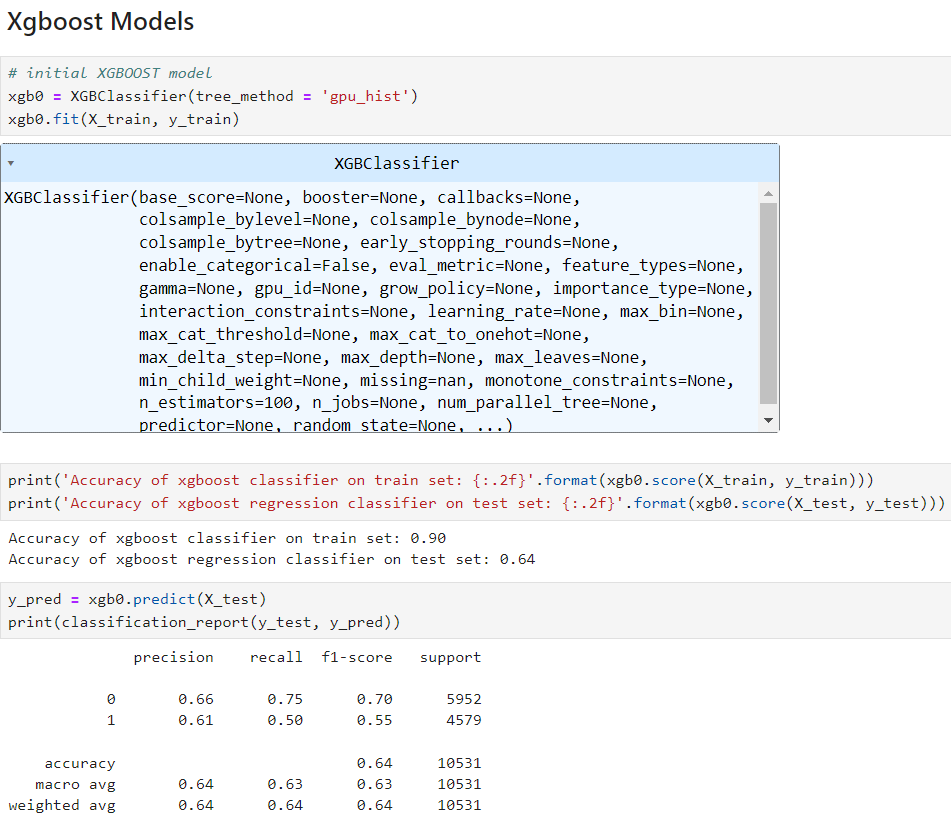
Below, the area-under-the-curve (AUC) score was calculated using the test set and the receiver operating characteristic ROC was plotted. This graph will be used later to compare against the final XGBoost model.





The ROC curve is larger than the unskilled (0.50) line which signifies that the model is slightly a little bit better at predicting an event than flipping a standard coin.

The initial XGBoost model that was trained had an accuracy of 0.90 on the training set and 0.64 on the test set. The initial model did not generalize well and it perhaps overfitted.



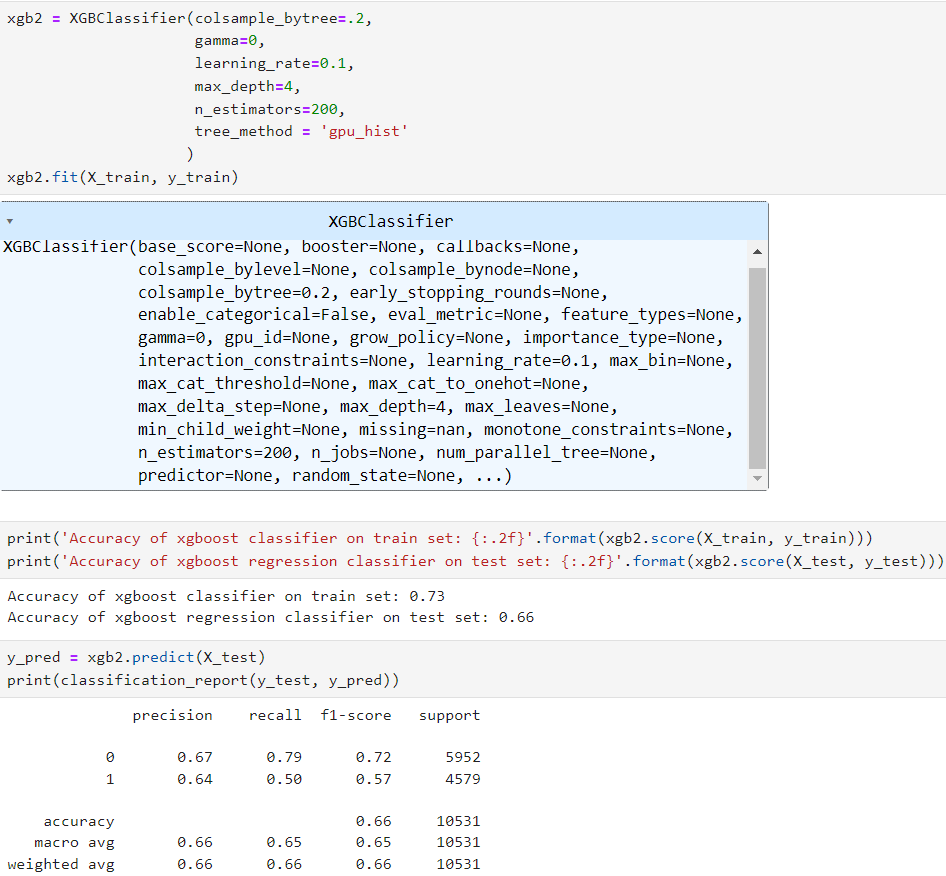
To remedy overfitting, we created a pipeline that utilized grid search again to determine the best parameter for XGBoost.



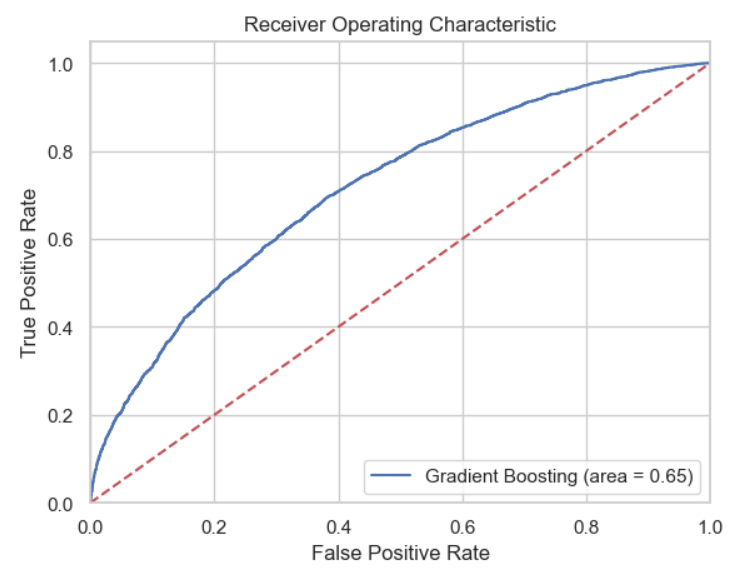
This time around, the accuracy is 0.69 on the train set and 0.67 on the test set. Although the accuracy is not stellar, the model generalized well and did not overfit.



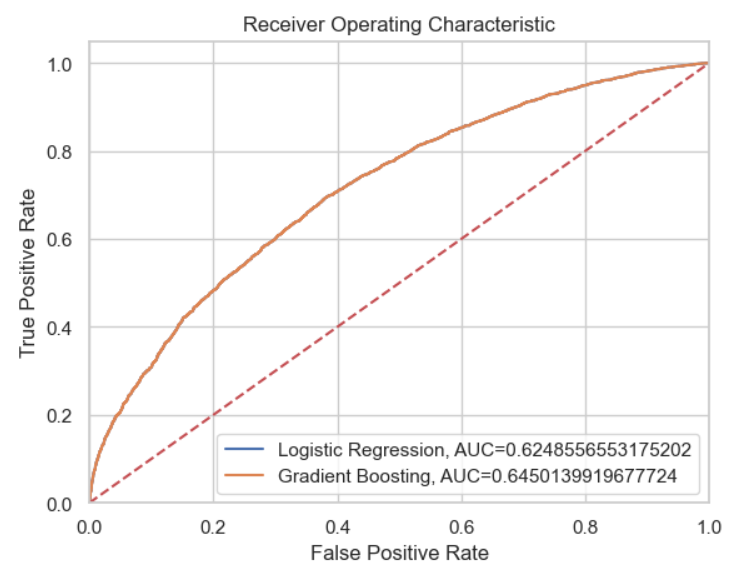
The best parameters were used in the final XGBoost model.



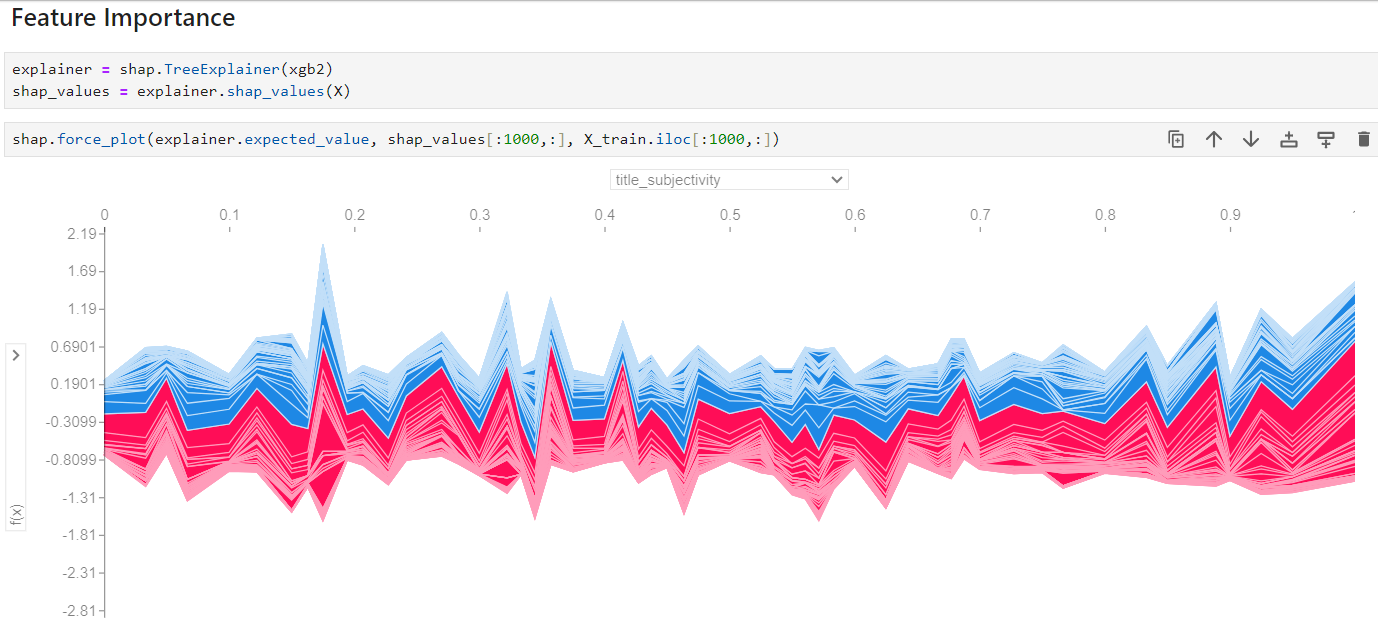
The final XGBoost model sports an accuracy of 0.73 on the train set and 0.66 on the test set. The AUC was calculated and ROC plotted.



When the best logistic regression model and the best XGBoost were plotted together, there was no indication that the two models were different at all. However, the AUC says otherwise. Nevertheless, XGBoost saw an improvement over logistic regression by one point in accuracy and two points in AUC score.

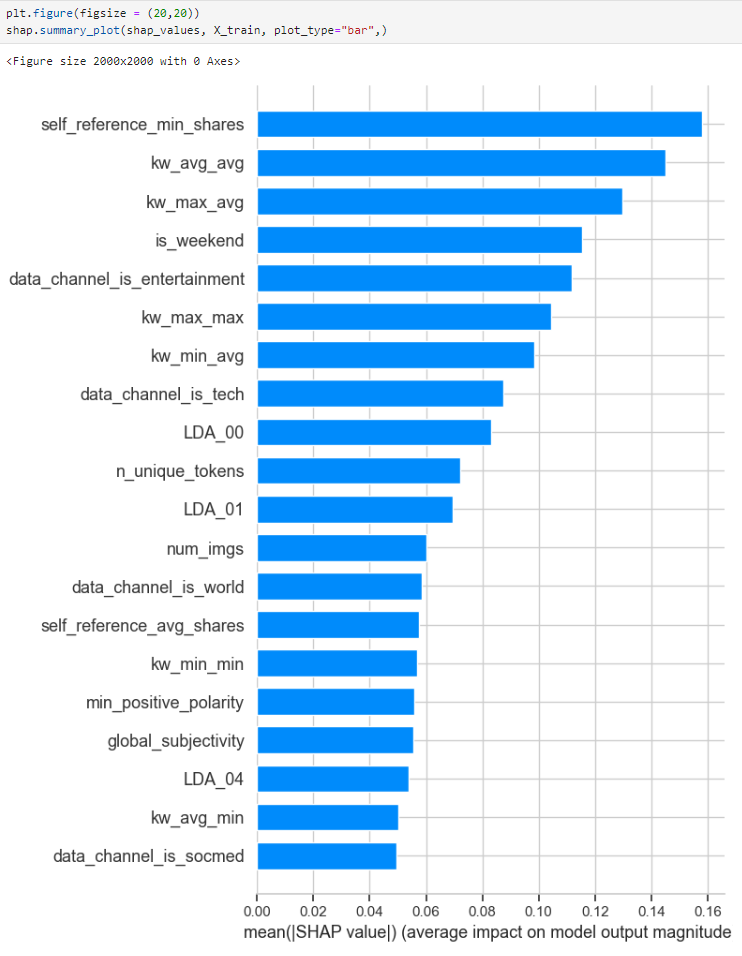


Now, feature importance will be extracted to determine the factors that contribute to an online news article’s popularity. For this purpose, the author decided to use the SHAP package to extract feature importance because those ranked by XGBoost were inconsistent. In “Interpretable Machine Learning with XGBoost,” Lundberg found that “feature importance orderings are very different for each of the three options provided by XGBoost” (2018). Therefore, SHAP method will be used to extract feature importance.

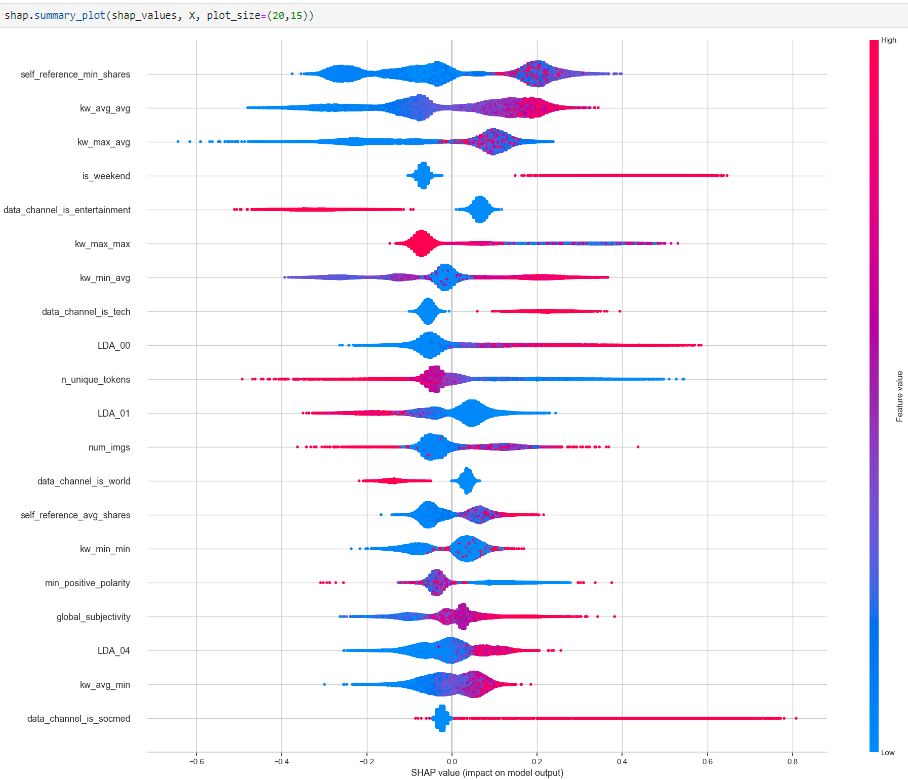


Above, we use the shap package to extract feature importance. Below, the shap package helps in extracting the most important features of our dataset. In relation to the final XGBoost model, the most important features are:

1. self\_reference\_min\_shares
2. kw\_avg\_avg
3. kw\_max\_avg
4. is\_weekend
5. data\_channel\_is\_entertainment



In the graph below, every article has one dot on each row. The x position of the dot is the impact of that feature on the model’s prediction for the article, and the color of the dot represents the value of that feature for the article. Dots that do not fit on the row pile up to show density (Lundberg, 2018).



E – Data Summary and Implications

The research question of this study is “Can gradient boosting be constructed based solely on the research data?” Based on the analysis using XGBoost, a gradient boosting model can be made from the Online News Popularity dataset with an accuracy of more than 65% and an AUC score of more than 60%. Some variables seem to have a stronger influence on the target variable as shown in the listing of mean SHAP values.

One limitation of this study is that the original dataset only included articles from one website (Mashable.com). The prevalence of popular articles in tech, business, and entertainment reflects the niched demographic of Mashable’s distribution. A sample containing articles from all sorts of publications would make a better dataset that could generalize better in predicting previously unseen articles.

Based on this study, one recommendation that can be made is to pay particular attention to the kind of articles that are published during the weekend. Even though the number of articles published on the weekend is less than those published during the week, the study shows that articles published during the weekend are more likely to be popular than not. This phenomenon surely warrants more investigation to determine the reason why.

The author of this study submits the following recommendation as course of actions for future studies:

* Use of XGBoost regression to predict the number of social media shares instead of using a threshold value
* Clustering the articles based on channels or topic

Initially converting the number of social media shares into the categorical number of 0 or 1 presented the possibility of information loss. Predicting the number of social media shares could possibly yield better results. In addition, leveraging clustering algorithms could also yield more insights about the different segments within the publication’s reader base.

# F – Sources

* Chen, Tianqi. (n.d.). Tianqi Chen. Retrieved August 31, 2023, from https://tqchen.com.
* Fernandes, Kelwin, Vinagre, Pedro, Cortez, Paulo, and Sernadela, Pedro. (2015). Online News Popularity. UCI Machine Learning Repository. https://doi.org/10.24432/C5NS3V
* How XGBoost Works. (n.d.). Amazon Sagemaker. Retrieved August 31, 2023, from https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html
* Lundberg, Scott. (2018, April 17). Interpretable Machine Learning with XGBoost. Towards Data Science. Retrieved from <https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27>.
* Mueller, Alex. (2018, March 24). 5 reasons why jupyter notebooks suck. Towards Data Science. Retrieved September 1, 2023, from https://towardsdatascience.com/5-reasons-why-jupyter-notebooks-suck-4dc201e27086
* Suginoo, Michio. (2022, October 20). XGBoost: its Genealogy, its Architectural Features, and its Innovation. Towards Data Science. Retrieved August 31, 2023, from https://towardsdatascience.com/xgboost-its-genealogy-its-architectural-features-and-its-innovation-bf32b15b45d2.
* Uddin, Md. Taufeeq (2018). Predicting the Popularity of Online News from Content Metadata. Retrieved August 29, 2023, from <https://github.com/krishnakartik1/onlineNewsPopularity/blob/master/Paper2/Predicting%20the%20Popularity%20of%20Online%20News%20from%20Content%20Metadata.pdf>

G – Appendix

# setting the random seed for reproducibility

import random

random.seed(493)

# for manipulating dataframes

import pandas as pd

import numpy as np

# for statistical testing

from scipy import stats

# for modeling

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import Pipeline

from sklearn.feature\_selection import SelectKBest, f\_classif

from sklearn.model\_selection import KFold

from sklearn import metrics

import statsmodels.api as sm

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve

from sklearn.metrics import accuracy\_score

from sklearn.metrics import make\_scorer

import xgboost as xgb

from xgboost import XGBClassifier

import shap

# for visualizations

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set(style="whitegrid")

# to print out all the outputs

from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast\_node\_interactivity = "all"

# set display options

import warnings

warnings.filterwarnings('ignore')

pd.set\_option('display.max\_columns', None)

pd.set\_option('display.max\_rows', None)

pd.set\_option('display.max\_colwidth', None)

# print the JS visualization code to the notebook

shap.initjs()

THRESHOLD = 1400

ALPHA = 0.05

# Read a csv file

df = pd.read\_csv('../data/in/OnlineNewsPopularity.csv')

df.head()

df.info()

df.shape

def show\_missing(df):

"""

Takes a dataframe and returns a dataframe with stats

on missing and null values with their percentages.

"""

null\_count = df.isnull().sum()

null\_percentage = (null\_count / df.shape[0]) \* 100

empty\_count = pd.Series(((df == ' ') | (df == '')).sum())

empty\_percentage = (empty\_count / df.shape[0]) \* 100

nan\_count = pd.Series(((df == 'nan') | (df == 'NaN')).sum())

nan\_percentage = (nan\_count / df.shape[0]) \* 100

dfx = pd.DataFrame({'num\_missing': null\_count, 'missing\_percentage': null\_percentage,

'num\_empty': empty\_count, 'empty\_percentage': empty\_percentage,

'nan\_count': nan\_count, 'nan\_percentage': nan\_percentage})

return dfx

show\_missing(df)

df.columns

for col in df.columns:

df = df.rename(columns={col:(col.strip(' '))})

df = df.rename(columns={'self\_reference\_avg\_sharess':'self\_reference\_avg\_shares'})

df.shape

df = df.drop\_duplicates(keep = False)

df.shape

# creates a new column for the new target variable and non-descriptive column

df['target'] = np.where(df['shares'] > 1400, int(1), int(0))

df = df.drop(columns=['url', 'timedelta'])

df.to\_csv('../data/out/online\_news\_popularity\_clean.csv', index=False)

df.head()

df.info()

df.describe()

sns.countplot(x='target', data=df)

token\_cols = ['n\_tokens\_title', 'n\_tokens\_content', 'n\_unique\_tokens', 'n\_non\_stop\_words', 'n\_non\_stop\_unique\_tokens', 'average\_token\_length', 'num\_keywords']

links\_cols = ['num\_hrefs', 'num\_self\_hrefs']

media\_cols = ['num\_imgs', 'num\_videos']

channel\_cols = ['data\_channel\_is\_lifestyle', 'data\_channel\_is\_entertainment', 'data\_channel\_is\_bus', 'data\_channel\_is\_socmed', 'data\_channel\_is\_tech', 'data\_channel\_is\_world']

kw\_cols = ['kw\_min\_min', 'kw\_max\_min', 'kw\_avg\_min', 'kw\_min\_max', 'kw\_max\_max', 'kw\_avg\_max', 'kw\_min\_avg', 'kw\_max\_avg', 'kw\_avg\_avg']

self\_ref\_cols = ['self\_reference\_min\_shares', 'self\_reference\_max\_shares', 'self\_reference\_avg\_shares']

week\_cols = ['weekday\_is\_monday', 'weekday\_is\_tuesday', 'weekday\_is\_wednesday', 'weekday\_is\_thursday', 'weekday\_is\_friday', 'weekday\_is\_saturday', 'weekday\_is\_sunday']

topic\_cols = ['LDA\_00', 'LDA\_01', 'LDA\_02', 'LDA\_03', 'LDA\_04']

global\_cols = ['global\_subjectivity', 'global\_sentiment\_polarity', 'global\_rate\_positive\_words', 'global\_rate\_negative\_words']

local\_cols = ['rate\_positive\_words', 'rate\_negative\_words', 'avg\_positive\_polarity', 'min\_positive\_polarity', 'max\_positive\_polarity', 'avg\_negative\_polarity', 'min\_negative\_polarity', 'max\_negative\_polarity']

title\_cols = ['title\_subjectivity', 'title\_sentiment\_polarity', 'abs\_title\_subjectivity', 'abs\_title\_sentiment\_polarity']

all\_columns = ['token\_cols', 'links\_cols', 'media\_cols', 'channel\_cols', 'kw\_cols', 'self\_ref\_cols',

'weekday\_cols', 'weekend\_cols', 'topic\_cols', 'global\_cols', 'local\_cols', 'title\_cols']

def viz\_box(df, col):

sns.boxplot(df[col], orient="h")

plt.title(str(col))

plt.show()

viz\_box(df, 'shares')

percentile25 = df['shares'].quantile(0.25)

percentile75 = df['shares'].quantile(0.75)

print("75th quartile: ", percentile75)

print("25th quartile: ", percentile25)

iqr = percentile75 - percentile25

upper\_bound = percentile75 + 1.5 \* iqr

lower\_bound = percentile25 - 1.5 \* iqr

df = df[df['shares'] < upper\_bound]

df = df[df['shares'] > lower\_bound]

print(len(df))

viz\_box(df, 'shares')

continuous\_cols = token\_cols + links\_cols + kw\_cols + self\_ref\_cols + global\_cols + local\_cols + title\_cols

for col in continuous\_cols:

viz\_box(df, col)

unpopular\_df = df[df['shares'] < THRESHOLD ]

popular\_df = df[df['shares'] >= THRESHOLD ]

popular\_df[continuous\_cols].hist(figsize=(20,20))

plt.show()

unpopular\_df[continuous\_cols].hist(figsize=(20,20))

plt.show()

corr = popular\_df[continuous\_cols].corr()

fig = plt.figure(figsize = (20,20))

sns.heatmap(corr, vmax = .8, square = True)

plt.show()

corr = unpopular\_df[continuous\_cols].corr()

fig = plt.figure(figsize = (20,20))

sns.heatmap(corr, vmax = .8, square = True)

plt.show()

# Adapted from

# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-by-side-dodged

# Numbers of pairs of bars you want

N = 7

# Data on X-axis

# Specify the values of blue bars (height)

popular\_week = popular\_df[week\_cols].sum().values

# Specify the values of orange bars (height)

unpopular\_week = unpopular\_df[week\_cols].sum().values

# Position of bars on x-axis

ind = np.arange(N)

# Figure size

plt.figure(figsize=(12,5))

# Width of a bar

width = 0.3

# Plotting

plt.bar(ind, popular\_week , width, label='Popular')

plt.bar(ind + width, unpopular\_week, width, label='Unpopular')

plt.xlabel('Days of the Week')

plt.ylabel('Count of News Articles')

plt.title('Count of Popular and Unpopular News Articles over Days of the Week')

# xticks()

# First argument - A list of positions at which ticks should be placed

# Second argument - A list of labels to place at the given locations

plt.xticks(ind + width / 2, ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'))

# Finding the best position for legends and putting it

plt.legend(loc='best')

plt.show()

# Adapted from

# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-by-side-dodged

# Numbers of pairs of bars you want

N = 7

# Data on X-axis

# Specify the values of blue bars (height)

popular\_week = popular\_df[week\_cols].sum().values

# Specify the values of orange bars (height)

unpopular\_week = unpopular\_df[week\_cols].sum().values

# Position of bars on x-axis

ind = np.arange(N)

# Figure size

plt.figure(figsize=(12,5))

# Width of a bar

width = 0.3

# Plotting

plt.bar(ind, popular\_week , width, label='Popular')

plt.bar(ind + width, unpopular\_week, width, label='Unpopular')

plt.xlabel('Days of the Week')

plt.ylabel('Count of News Articles')

plt.title('Count of Popular and Unpopular News Articles over Days of the Week')

# xticks()

# First argument - A list of positions at which ticks should be placed

# Second argument - A list of labels to place at the given locations

plt.xticks(ind + width / 2, ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'))

# Finding the best position for legends and putting it

plt.legend(loc='best')

plt.show()

# Adapted from

# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-by-side-dodged

# Numbers of pairs of bars you want

N = 6

# Data on X-axis

# Specify the values of blue bars (height)

popular\_week = popular\_df[channel\_cols].sum().values

# Specify the values of orange bars (height)

unpopular\_week = unpopular\_df[channel\_cols].sum().values

# Position of bars on x-axis

ind = np.arange(N)

# Figure size

plt.figure(figsize=(12,5))

# Width of a bar

width = 0.3

# Plotting

plt.bar(ind, popular\_week , width, label='Popular')

plt.bar(ind + width, unpopular\_week, width, label='Unpopular')

plt.xlabel('Channels')

plt.ylabel('Count of News Articles')

plt.title('Count of Popular and Unpopular News Articles over Days of the Week')

# xticks()

# First argument - A list of positions at which ticks should be placed

# Second argument - A list of labels to place at the given locations

plt.xticks(ind + width / 2, ('Lifestyle', 'Entertainment', 'Business', 'Social Media', 'Tech', 'World'))

# Finding the best position for legends and putting it

plt.legend(loc='best')

plt.show()

# Adapted from

# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-by-side-dodged

# Numbers of pairs of bars you want

N = 5

# Data on X-axis

# Specify the values of blue bars (height)

popular\_week = popular\_df[topic\_cols].sum().values

# Specify the values of orange bars (height)

unpopular\_week = unpopular\_df[topic\_cols].sum().values

# Position of bars on x-axis

ind = np.arange(N)

# Figure size

plt.figure(figsize=(12,5))

# Width of a bar

width = 0.3

# Plotting

plt.bar(ind, popular\_week , width, label='Popular')

plt.bar(ind + width, unpopular\_week, width, label='Unpopular')

plt.xlabel('LDA Topic')

plt.ylabel('Count of News Articles')

plt.title('Count of Popular and Unpopular News Articles over LDA Topics')

# xticks()

# First argument - A list of positions at which ticks should be placed

# Second argument - A list of labels to place at the given locations

plt.xticks(ind + width / 2, ('LDA\_00', 'LDA\_01', 'LDA\_02', 'LDA\_03', 'LDA\_04'))

# Finding the best position for legends and putting it

plt.legend(loc='best')

plt.show()

ttest\_same = []

ttest\_diff = []

for column in continuous\_cols:

result = stats.ttest\_ind(popular\_df[column], unpopular\_df[column])[1]

if result > ALPHA:

interpretation = 'insignificant - SAME'

ttest\_same.append(column)

else:

interpretation = 'significant - DIFFERENT'

ttest\_diff.append(column)

print(result, '-', column, ' - ', interpretation)

shap\_yesn = []

shap\_notn = []

for column in continuous\_cols:

stat, result = stats.shapiro(df[column])

if result > ALPHA:

interpretation = 'insignificant - NORMAL'

shap\_yesn.append(column)

else:

interpretation = 'significant - NOT NORMAL'

shap\_notn.append(column)

print(result, '-', column, ' - ', interpretation)

final\_df = df.drop(columns=['shares'])

X = final\_df.loc[:, final\_df.columns != 'target']

y = final\_df.loc[:, final\_df.columns == 'target']

# Train Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=493)

logreg0 = LogisticRegression()

logreg0.fit(X\_train, y\_train)

print('Accuracy of logistic regression on train set: {:.2f}'.format(logreg0.score(X\_train, y\_train)))

print('Accuracy of logistic regression on test set: {:.2f}'.format(logreg0.score(X\_test, y\_test)))

y\_pred = logreg0.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

# parameter grid

parameters = {

'penalty' : ['l1','l2'],

'C' : np.logspace(-3,3,7),

'solver' : ['newton-cg', 'lbfgs', 'liblinear'],

}

logreg1 = LogisticRegression()

clf = GridSearchCV(logreg1, # model

param\_grid = parameters, # hyperparameters

scoring='accuracy', # metric for scoring

cv=10) # number of folds

clf.fit(X\_train,y\_train)

print("Tuned Hyperparameters :", clf.best\_params\_)

print("Logistic Regression) Accuracy :",clf.best\_score\_)

logreg2 = LogisticRegression(C = 10,

penalty = 'l1',

solver = 'liblinear')

logreg2.fit(X\_train,y\_train)

print('Accuracy of logistic regression classifier on train set: {:.2f}'.format(logreg2.score(X\_train, y\_train)))

print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg2.score(X\_test, y\_test)))

y\_pred = logreg2.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

logreg\_roc\_auc = roc\_auc\_score(y\_test, logreg2.predict(X\_test))

fpr, tpr, thresholds = roc\_curve(y\_test, logreg2.predict\_proba(X\_test)[:,1])

plt.figure()

plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logreg\_roc\_auc)

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

# initial XGBOOST model

xgb0 = XGBClassifier(tree\_method = 'gpu\_hist')

xgb0.fit(X\_train, y\_train)

print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb0.score(X\_train, y\_train)))

print('Accuracy of xgboost classifier classifier on test set: {:.2f}'.format(xgb0.score(X\_test, y\_test)))

y\_pred = xgb0.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

pipe = Pipeline([

('fs', SelectKBest()),

('clf', xgb.XGBClassifier(objective='binary:logistic'))

])

# Define our search space for grid search

search\_space = [

{

'clf\_\_n\_estimators': [100, 200],

'clf\_\_learning\_rate': [0.1, 0.01],

'clf\_\_max\_depth': [3, 4, 5],

'clf\_\_colsample\_bytree': [0.1, 0.2],

'clf\_\_gamma': [0],

'clf\_\_tree\_method': ['gpu\_hist'],

'fs\_\_score\_func': [f\_classif],

'fs\_\_k': [10],

}

]

# Define cross validation

kfold = KFold(n\_splits=10)

# AUC and accuracy as score

scoring = {'AUC':'roc\_auc', 'Accuracy':make\_scorer(accuracy\_score)}

# Define grid search

grid = GridSearchCV(

pipe,

param\_grid=search\_space,

cv=kfold,

scoring=scoring,

refit='AUC',

verbose=1,

n\_jobs=-1

)

# Fit grid search

xgb1 = grid.fit(X\_train, y\_train)

print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb1.score(X\_train, y\_train)))

print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb1.score(X\_test, y\_test)))

y\_pred = xgb1.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

print(xgb1.best\_params\_)

xgb2 = XGBClassifier(colsample\_bytree=.2,

gamma=0,

learning\_rate=0.1,

max\_depth=4,

n\_estimators=200,

tree\_method = 'gpu\_hist'

)

xgb2.fit(X\_train, y\_train)

print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb2.score(X\_train, y\_train)))

print('Accuracy of xgboost classifier classifier on test set: {:.2f}'.format(xgb2.score(X\_test, y\_test)))

y\_pred = xgb2.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

xgbc\_roc\_auc = roc\_auc\_score(y\_test, xgb2.predict(X\_test))

fpr, tpr, thresholds = roc\_curve(y\_test, xgb2.predict\_proba(X\_test)[:,1])

plt.figure()

plt.plot(fpr, tpr, label='Gradient Boosting (area = %0.2f)' % xgbc\_roc\_auc)

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

#set up plotting area

plt.figure(0).clf()

plt.plot(fpr,tpr,label="Logistic Regression, AUC=" + str(logreg\_roc\_auc))

plt.plot(fpr,tpr,label="Gradient Boosting, AUC=" + str(xgbc\_roc\_auc))

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

explainer = shap.TreeExplainer(xgb2)

shap\_values = explainer.shap\_values(X)

shap.force\_plot(explainer.expected\_value, shap\_values[:1000,:], X\_train.iloc[:1000,:])

plt.figure(figsize = (20,20))

shap.summary\_plot(shap\_values, X\_train, plot\_type="bar",)

shap.summary\_plot(shap\_values, X, plot\_size=(20,20))

for col in X\_train.columns:

shap.dependence\_plot(col, shap\_values, X)

print('Successful run!')