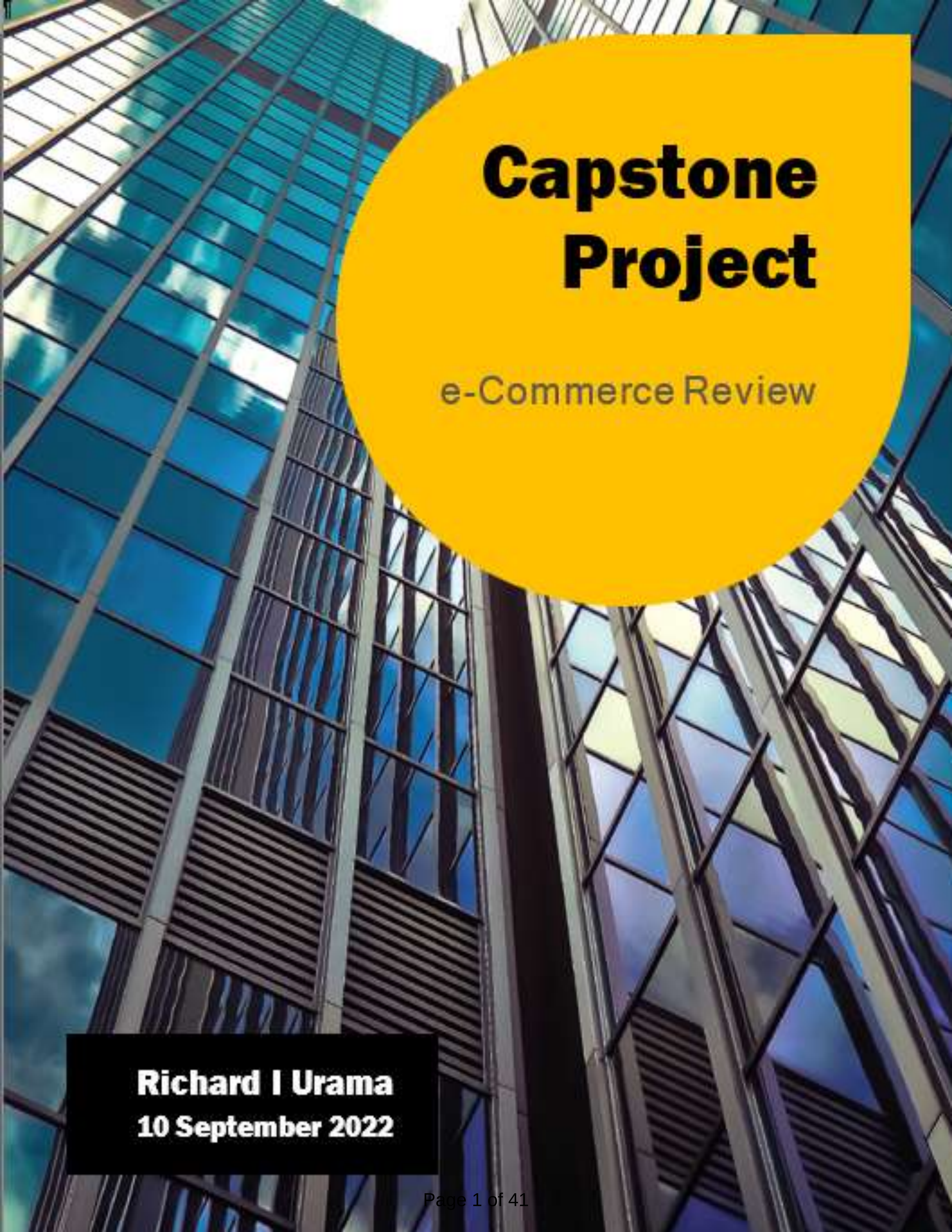


Capstone Project

e-Commerce Review

Richard I Urama

10 September 2022

Table of Contents

1 PROBLEM STATEMENT

An online e-commerce company is having significant operations challenges regarding its customer relations which is adversely affecting its bottom.

As the company's Data Analyst, you have been tasked to analyze its customer reviews data for various products and come up with report that classifies the products based on the customer reviews. The report should:

1. Find various trends and patterns in the reviews data, create useful insights that best describe the product quality.
2. Classify each review based on the sentiment associated with the same

.

2 Project Objective

Machine learning (ML) as coined by Arthur Samuel (Checkers AI, 1959) has indeed come of age and has become one of the important components of the growing field of data science. The underlying idea in ML is to use statistical methods to 'train' algorithms for the purpose of classifications and predictions whose aim is to uncover key insights from the deluge of data at the disposal of most businesses today. The gained insights will ideally provide the decision makers in the industry the ability to make better informed decisions that will improve the growth metrics of their businesses.

In this project, the customer review data generated by an e-commerce company will be analyzed to gain useful insights from the up-votes and/or down-votes of its teaming customers. It is hoped that the uncovered insights and the recommendations that will be generated from the analysis will greatly assist the company in refocusing its operations to respond to the needs of her customers.

Natural Language Processing (NLP) will be employed for the analysis. NLP helps us understand the structure and meaning of human language through the analysis of syntax, semantics, pragmatics, and morphology The linguistic knowledge is transformed structured rule-based machine learning algorithm that is able to predict specific outcomes. NLP can be deployed in two major areas, namely: *Natural Language Understanding* and *Natural Language Generation*. Based on the objective of this project, our focus will be on Natural Language Understanding

.

3 DATA DESCRIPTION

This dataset contains over 568k of consumer reviews on different products of the company.

Feature Name	Description	Details
Id	Record ID	Range: 1 - 568K
ProductId	Product ID	74,258 unique values
UserId	User ID who posted the review	256,059 unique values
ProfileName	Profile name of the User	218,418 unique values
HelpfullnessNumerator	Numerator of the helpfulness of the review	Range: 0 - 866
HelpfullnessDenominator	Denominator of the helpfulness of the review	Range: 0 - 923
Score	Product Rating	Range: 1 - 5
Time	Review time in timestamp	8,662 records
Summary	Summary of the review	295,744 unique values
Text	Actual text of the review	393,579 unique values

Data Description: The data description showing the total count, mean, standard deviation(std), minimum, etc. for the numeric features is as shown in the Table

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	568411.000000	568411.000000	568411.000000	568411.000000	5.684110e+05
mean	284227.440964	1.743874	2.227876	4.183309	1.296261e+09
std	164099.020907	7.636781	8.288752	1.310368	4.803792e+07
min	1.000000	0.000000	0.000000	1.000000	9.393408e+08
25%	142114.500000	0.000000	0.000000	4.000000	1.271290e+09
50%	284224.000000	0.000000	1.000000	5.000000	1.311120e+09
75%	426341.500000	2.000000	2.000000	5.000000	1.332720e+09
max	568454.000000	866.000000	923.000000	5.000000	1.351210e+09

3 DATA DESCRIPTION

Dataset types:

	Types
Id	int64
ProductId	object
UserId	object
ProfileName	object
HelpfulnessNumerator	int64
HelpfulnessDenominator	int64
Score	int64
Time	int64
Summary	object
Text	object

Percentage of null values in the dataset features

	% Null
Id	0.00000
ProductId	0.00000
UserId	0.00000
ProfileName	0.00281
HelpfulnessNumerator	0.00000
HelpfulnessDenominator	0.00000
Score	0.00000
Time	0.00000
Summary	0.00475
Text	0.00000

Cor-relation of the features:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
Id	1.000000	0.001227	0.000770	0.010706	0.007912
HelpfulnessNumerator	0.001227	1.000000	0.974689	-0.032590	-0.154818
HelpfulnessDenominator	0.000770	0.974689	1.000000	-0.097986	-0.173289
Score	0.010706	-0.032590	-0.097986	1.000000	-0.062760
Time	0.007912	-0.154818	-0.173289	-0.062760	1.000000

4 DATA PREPROCESSING STEPS AND INSPIRATION

The preprocessing of the data included the following steps:

1. Loading the eCommerce customer review data
2. Inspect for missing data and take necessary action as per the following:
    a. Remove the rows with missing value if their number is insignificant
    b. Replace the missing values with the mean or median if the feature is numeric
3. Calculate the rate of helpfulness by dividing the HelpfulnessNumerator by HelpfulnessDenominator and where the HelpfulnessDenominator is zero, assign a value of -1 to signify negative or no-Helpfulness
4. Create bins and labels for the Helpfulness as follows:
    a. Bins: [-1, 0, 0.2, 0.4, 0.6, 0.8, 1.0]
    b. Labels: ['Null', '0-20%', '20-40%, '40-60%', '60-80%', '80-100%']
5. Convert the the feature Time to YYYY-MM-DD time record
6. Extract the aggregate count of the feature values in each bin based on Score value and store in df_summary
7. Create a pivot Table of the df_summary data for better appreciation/understanding
8. Create a heatmap of the df_summary for better visualization.
9. Convert the unique Scores [1, 2, 3, 4, 5] into two categories such that:
    a. 1 ➔ 4 or 5
    b. 0 ➔ 1 or 2
    c. 3 ➔ Neutral

    ➔ Which results in score_dict: {1:0, 2:0, 4:1, 5:1}
10. Since a vote of 3 is considered neutral, it will be excluded from the model training dataset:
    a. The Text feature of the data (excluding Score of 3) is stored in X and the Score feature (maped according to the established dictionary → score_dict) is stored in y, the dependent variable.
11. Using module CountVectorizer, the Text feature is converted to vectors and stored in X_vectorized.

4 DATA PREPROCESSING STEPS AND INSPIRATION

12. Using the train_test_split from sklearn model selection module, extract the X_train, X_test, y_train, y_test for the Logistic regression model with data split of 80:20 and random_state of 42.
13. Train the model with LogisticRegression() and obtain predictions and accuracy
14. Although the accuracy result from the LogisticRegression() was okay, the assignment of coefficients to the significant words was not reasonable.
15. To correct the anomaly, Term Frequency — Inverse Document Frequency (TF-IDF), TfidfVectorizer was applied for feature extraction. The resulting accuracy was pratically the same but the coefficient assignment to significant words are now more reasonable.
16. It is observed that the dependent feature y is highly skewed to Score of 5 as shown the adjacent Table-
17. RandomOverSampler from imblearn was used to fit the train/test data and the resultant was well balanced (see Table below)

Before OverSampling	**After OverSampling**
Score:------->y_train: Total _Count 1.0 121375 0.0 2165	Score:------->y_train: Total _Count 1.0 151719 0.0 151719

18. To improve our model, We will employ GridSearchCV which is a process of performing hyperparameter tuning in order to determine the optimal values for a given model. GridSearchCV loops through predefined hyperparameters and fit the estimator (model) on the training set. The result is that the best parameters from the listed hyperparameters is selected. With the optimal the model accuracy improved from 93.6 to 98.7% with recall rate of 99% for up-votes and 100% for down-votes.

.

5 Choosing the Algorithm for the Project

The choice for my algorithm for the project was dictated by the project objective which is to analyze customer reviews and predict the direction of their votes in either up-vote (1) or down-vote (0). This is therefore a classical classification problem which Logistic Regression algorithm handles very well with proven accuracy of better than 90% in most cases.

Logistic Regression which I adopted for the project is a linear regression algorithm for Machine Learning specifically engineered to predict the probability of binary categorical dependent features consisting of either [Yes or No], [Pass or Fail] vote and which could be translated to 0 or 1.

.

6 ASSUMPTIONS

The Logistic Regression algorithm is based on some assumptions which include the following:

1. **Sample size**: It needs large sample size for reliable prediction which this project satisfies with over 586K records. The empirical rule of thumb is minimum 20 samples for every independent variable.
2. **Linear relationship**: The independent variables (X) have a linear relation with the dependent variable (y). It's assumed that each customer voted individually and was not influenced.
3. **Multicollinearity** There should be minimal relationship (multicollinearity) between observations the independent variables.
4. **Autocorrelation**: There should be minimal or no relationship between data points of any given feature which implies that all data points of an independent feature in a sample should be independent of each other.

7 MODEL EVALUATION AND TECHNIQUE

The model evaluation technique includes the following steps:

1. Import libraries
2. Load dataset
3. Perform EDA
4. Perform data cleaning by removing null values
5. Prepare data by calculating Helpfullness rate by dividing the HelpfulnessNumerator by HelpfulnessDenominator.
6. Drop rows with scores of 3 which are considered neutral votes
7. Replaced null values with -1. indicating that there was no vote.
   - Creating bins to segment the Helpfullness rate:
   - ['Empty', '0-20%', '20-40%', '40-60%', '60-80%', '80-100%']
8. Create Pivot Table
9. Plot heatmap of the pivot Table, for better visualizations
10. Convert the Scores into 2 categories:
    - 0 (negative or down votes) for scores of 1 or 2 and
    - 1 (positive or up votes) for scores of 4 or 5
11. Create a dictionary of the scores: score_dict = { 1:0, 2:0, 4:1,5:1}
12. Using the map() function, map the scores to the dataset using the scores dictionary from line 10.

7 MODEL EVALUATION AND TECHNIQUE

13. **Model Training**: Two approaches were implemented to find the best accuracy:

    I. **First Approach**:

    a. **NLP**: CountVectorizer() to transform the Text column (X) into vectors based on the number of occurrences of the word in the text.

    b. **Split data** (X_vectorized, y) into train & test

    c. **Model training** with LogisticRegression

    d. **Predictions**

    e. **Compute model accuracy** → *model accuracy was 93.5%*

14. Two observations were made after the first approach: (1) certain significant coefficients didn't make much sense, and (2) the train data was skewed in favor of the positive votes.

15. A second approach was implement to correct the anomalies.

    I. **Second Approach:**

    a. **NLP:** TfidfVectorizer()

    b. **Split data** (X_vectorized, y) into train & test

    c. **Apply RandomOverSampler()** to balance train data

    d. **Apply** GridSearchCV with LogisticRegression to perform hyperparameter tuning

    e. **Train the Model**

    f. **Predictions**

    g. **Model Accuracy →** model accuracy was 98.7%

    h. **Obtain Classification Report**

8 INFERENCES FROM THE PROJECT

1. The model accuracy was 98.7%
2. The following can be inferred from the classification report shown below:

Model Accuracy: 98.73%

Classification Report:

	precision	recall	f1-score	support
0.0	0.58	1.00	0.73	542
1.0	1.00	0.99	0.99	30344
accuracy			0.99	30886
macro avg	0.79	0.99	0.86	30886
weighted avg	0.99	0.99	0.99	30886

   a. **Precision** of the up-votes of 1.0 implies that 100% of the customers who had positive reviews(up-vote) were correctly identified. This is also supported by the recall (percentage of correct positive predictions) and f1-score (the weighted harmonic mean of precision and recall) which are each 99%.
   b. However, only 58% of the down-votes were correctly identified which means that the model is highly skewed towards the positive reviews (up-votes).
   c. **Support**: This is the number of actual occurrences of the class in the dataset. The support values of **542 (1,75%)** for the negative reviews(down-votes) when compared to **30,344(98.75%)** for the positive reviews demonstrates the skewness of the dataset towards positive reviews.
   d. It is also noted that more than half of the reviews had no vote.
   e. The model result shows that many customers had positive reviews (opinions) about the company.

9 Future Possibilities

This project has exposed us to the power of **Natural language processing** (NLP) technology which was able to analyze over five hundred thousand consumer review records to come up with a conclusive direction of the majority of the opinion votes (positive or negative) in the report. This task would ordinarily be almost impossible and very time consuming for humans to achieve.

NLP, a subfield of Artificial Intelligence (AI) enables machines to efficiently understand and interpret human languages. The future possibilities of this technology are endless as is demonstrated by the cutting-edge AI applications based on the technology currently in the market. These applications include:

1. NLP-powered Digit Assistant: these applications can literarily communicate just like humans, interpret human languages and perform specific assigned tasks. The leading applications in this area currently are Siri (Apple), Ok Google (Google), Cortana (Microsoft), and Alexa (Amazon).

2. Interactive Voice Recognition (IVR): This is one of the significant examples of NLP systems in the industry today. It serves as gate-keeper or interphase between a company and its customers, receiving and analyzing calls and routing the calls if necessary to the appropriate person. Almost all major organizations have deployed IVR to collect data from callers, analyze and route the calls

The future of natural language processing in AI promises to be thrilling. It’s predicted that technology advancements in this area will be very exciting in light of innovations derived from the combinations of AI, NLP, Internet of Things (IoT), and ML. NLP technology will play a major role in the implementation of gesture and facial recognition apps, which scope is promising in the very near future.

10 CONCLUSION

The project provided a great opportunity to appreciate the power of NLP in ML which is able to extract meaningful insights from customer review data of over five hundred thousand records. This task ordinarily would have been inconceivable without NLP technology.

The ability to extract meaningful insights, in a timely manner, from hundreds of thousands or in some cases millions of customer review records is a game changer for service-oriented organizations and industries who now have prompt information regarding their customers reactions/feelings towards their goods and/or services enabling them to respond quickly making necessary corrections to ensure better customer service. Improved customer service will ultimately lead to increased profitability since happy customers are more likely to be repeat customers and will also attract new ones.

References

Links -

1. https://usmsystems.com/natural-language-processing/
2. Logistic Regression in Python - Real Python

Book -

1. NLP: The Essential Guide to Neuro-Linguistic Programming Paperback – February 12, 2013 **by** by NLP Comprehensive, Tom Dotz, and Tom Hoobyar
2. Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems 1st Edition by Sowmya Vajjala (Author), Bodhisattwa Majumder (Author), Anuj Gupta (Author), Harshit Surana
3. Machine Learning for Beginners Guide Algorithms: Supervised & Unsupervised Learning. Decision Tree & Random Forest Introduction (Artificial Intelligence Book 1)
4. Hands-On Machine Learning from Scratch: Develop a Deeper Understanding of Machine Learning Models by Implementing Them from Scratch in Python
5. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, 2nd Edition

Capstone Project - e-Commerce Company | Richard Urama

```
Import some necessary modules
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Helper Functions

In [3]:
```
def seconds_to_day(seconds):
    from datetime import date
    return date.fromtimestamp(seconds)
```

In [4]:
```
# Function to return rounded dbf

def round_val(data):
    for i,j in enumerate(data):
        data[i]=round(j,2)
    return (data)
```

In [5]:
```
def format_number(number):
    return("{:,}".format(number))
```

Step 1: Read & Understand The Input Dataset (Retail.csv)

In [6]:
```
df=pd.read_csv("Reviews.csv")
```

In [7]:
```
#pd.set_option('expand_frame_repr', False)                # keep print on the sam
df.head(3)
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominato
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	(
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [8]:
```
df.shape
```

Out[8]: (568454, 10)

In [9]:
```
# Examine some of the text data

for i in range(5):
    print(f"Review item {i+1}")
    print(f"SUmmary: {df['Summary'][i]}")
    print(f"Text    : {df['Text'][i]}\n")
```

```
Review item 1
SUmmary: Good Quality Dog Food
Text    : I have bought several of the Vitality canned dog food products and have
found them all to be of good quality. The product looks more like a stew than a
processed meat and it smells better. My Labrador is finicky and she appreciates
this product better than  most.

Review item 2
SUmmary: Not as Advertised
Text    : Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actu
ally small sized unsalted. Not sure if this was an error or if the vendor intend
ed to represent the product as "Jumbo".

Review item 3
SUmmary: "Delight" says it all
Text    : This is a confection that has been around a few centuries.  It is a lig
ht, pillowy citrus gelatin with nuts - in this case Filberts. And it is cut into
tiny squares and then liberally coated with powdered sugar.  And it is a tiny mo
uthful of heaven.  Not too chewy, and very flavorful.  I highly recommend this y
ummy treat.  If you are familiar with the story of C.S. Lewis' "The Lion, The Wi
tch, and The Wardrobe" - this is the treat that seduces Edmund into selling out
his Brother and Sisters to the Witch.

Review item 4
SUmmary: Cough Medicine
Text    : If you are looking for the secret ingredient in Robitussin I believe I
have found it.  I got this in addition to the Root Beer Extract I ordered (which
was good) and made some cherry soda.  The flavor is very medicinal.

Review item 5
SUmmary: Great taffy
Text    : Great taffy at a great price.  There was a wide assortment of yummy taf
fy.  Delivery was very quick.  If your a taffy lover, this is a deal.
```

In [10]:

```
# Check data types
df_types = df.dtypes
pd.DataFrame(df.dtypes.values, index=df.dtypes.index, columns=['Types'])
```

Out[10]:

	Types
Id	int64
ProductId	object
UserId	object
ProfileName	object
HelpfulnessNumerator	int64
HelpfulnessDenominator	int64
Score	int64
Time	int64
Summary	object
Text	object

In [11]:

```
df.isnull().sum()
```

Out[11]:

```
Id                         0
ProductId                  0
UserId                     0
ProfileName               16
HelpfulnessNumerator       0
HelpfulnessDenominator     0
Score                      0
Time                       0
Summary                   27
Text                       0
dtype: int64
```

In [12]:

```python
# Calculating the Missing Values % contribution in DF

df_null = round(df.isnull().sum()/df.shape[0]*100,5)
pd.DataFrame(df_null.values, index=df_null.index, columns=['% Null'])
```

Out[12]:

	% Null
Id	0.00000
ProductId	0.00000
UserId	0.00000
ProfileName	0.00281
HelpfulnessNumerator	0.00000
HelpfulnessDenominator	0.00000
Score	0.00000
Time	0.00000
Summary	0.00475
Text	0.00000

In [13]:

```python
# Rows with no missing ProfileNames & Summary (0.000028% & 0.000047) respectively
#        for our model, and can thus be removed from consideration without much cor
#        model prediction

df = df.dropna()
df.shape
```

Out[13]: (568411, 10)

In [14]:

```python
df.isnull().sum()
```

Out[14]:

```
Id                        0
ProductId                 0
UserId                    0
ProfileName               0
HelpfulnessNumerator      0
HelpfulnessDenominator    0
Score                     0
Time                      0
Summary                   0
Text                      0
dtype: int64
```

In [15]:

```python
df['Score'].value_counts()
```

Out[15]:

```
5    363111
4     80655
1     52264
3     42638
2     29743
Name: Score, dtype: int64
```

In [16]:
```
# check for duplicates
#   No duplicates found

df[df.duplicated()].shape[0]
```
Out[16]:
```
0
```

In [17]:
```
df.corr()
```
Out[17]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tir
Id	1.000000	0.001225	0.000760	0.010712	0.0079
HelpfulnessNumerator	0.001225	1.000000	0.974849	-0.032594	-0.1548
HelpfulnessDenominator	0.000760	0.974849	1.000000	-0.097808	-0.1730
Score	0.010712	-0.032594	-0.097808	1.000000	-0.0629
Time	0.007913	-0.154850	-0.173043	-0.062964	1.0000

In [18]:
```
df.describe()
```
Out[18]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	568411.000000	568411.000000	568411.000000	568411.000000	5.684110e+05
mean	284227.440964	1.743874	2.227876	4.183309	1.296261e+09
std	164099.020907	7.636781	8.288752	1.310368	4.803792e+07
min	1.000000	0.000000	0.000000	1.000000	9.393408e+08
25%	142114.500000	0.000000	0.000000	4.000000	1.271290e+09
50%	284224.000000	0.000000	1.000000	5.000000	1.311120e+09
75%	426341.500000	2.000000	2.000000	5.000000	1.332720e+09
max	568454.000000	866.000000	923.000000	5.000000	1.351210e+09

In [19]:
```
df.corr()['Score']
```
Out[19]:
```
Id                        0.010712
HelpfulnessNumerator     -0.032594
HelpfulnessDenominator   -0.097808
Score                     1.000000
Time                     -0.062964
Name: Score, dtype: float64
```

Step 2 : Data Cleansing

In [20]:
```
df.columns
```
Out[20]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

In [21]:

```python
Time=[]
seconds_list=np.array(df['Time'].values)

for second in seconds_list:
    Time.append(seconds_to_day(second))
pd.DataFrame(Time, columns=['Time']).head()
```

Out[21]:

	Time
0	2011-04-26
1	2012-09-06
2	2008-08-17
3	2011-06-12
4	2012-10-20

In [22]:

```python
# Time column in df to datatimestamp

df['Time']=Time
df.head()
```

Out[22]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominato
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

In [23]:
```
df['Time'].min()
```
Out[23]:
```
datetime.date(1999, 10, 7)
```

In [24]:
```
df['Time'].max()
```
Out[24]:
```
datetime.date(2012, 10, 25)
```

In [25]:
```
df['Time'].head()
```
Out[25]:
```
0    2011-04-26
1    2012-09-06
2    2008-08-17
3    2011-06-12
4    2012-10-20
Name: Time, dtype: object
```

In [26]:
```
df[df['HelpfulnessNumerator']==0].shape
```
Out[26]:
```
(303813, 10)
```

In [27]:
```
# define Helpfulness as a measure/rate of how helpful the particular rating was:
#        where 1 (100%) would signify vey helpful
#              0 not helpful
#
df['Helpfulness']=df['HelpfulnessNumerator']/df['HelpfulnessDenominator']
df['Helpfulness']
```
Out[27]:
```
0         1.0
1         NaN
2         1.0
3         1.0
4         NaN
         ...
568449    NaN
568450    NaN
568451    1.0
568452    1.0
568453    NaN
Name: Helpfulness, Length: 568411, dtype: float64
```

In [28]:
```
# For cases where Helpfulness is NAN, meaning the denominator ws zero or no helpfu
#    we will assign -1

df['Helpfulness']=df['Helpfulness'].fillna(-1)
print(format_number(df[df['Helpfulness']==-1].shape[0]))
```
```
270,039
```

In [29]:

```
df['Helpfulness']
```

Out[29]:

```
0         1.0
1        -1.0
2         1.0
3         1.0
4        -1.0
         ...
568449   -1.0
568450   -1.0
568451    1.0
568452    1.0
568453   -1.0
Name: Helpfulness, Length: 568411, dtype: float64
```

In [30]:

```
Helpfullness=df[['HelpfulnessNumerator','HelpfulnessDenominator','Helpfulness','Sc
                ]].sort_values(by='HelpfulnessDenominator',ascending=False)

Helpfullness.head()
```

Out[30]:

	HelpfulnessNumerator	HelpfulnessDenominator	Helpfulness	Score	Time
207712	844	923	0.914410	3	2009-09-07
190733	866	878	0.986333	5	2006-11-27
566779	808	815	0.991411	5	2009-12-13
235722	580	593	0.978078	1	2011-07-01
222937	491	569	0.862917	3	2008-05-31

In [31]:

```
Helpfullness.tail()
```

Out[31]:

	HelpfulnessNumerator	HelpfulnessDenominator	Helpfulness	Score	Time
256616	0	0	-1.0	2	2011-01-02
256615	0	0	-1.0	3	2011-01-02
256614	0	0	-1.0	2	2011-01-04
256613	0	0	-1.0	4	2011-01-10
568453	0	0	-1.0	5	2012-05-30

In [32]:
```python
Helpfull=Helpfullness[Helpfullness['HelpfulnessNumerator']>0 &
                      (Helpfullness['HelpfulnessNumerator'] <=
                    Helpfullness['HelpfulnessDenominator']) ]
Helpfull.head()
```

Out[32]:

	HelpfulnessNumerator	HelpfulnessDenominator	Helpfulness	Score	Time
207712	844	923	0.914410	3	2009-09-07
190733	866	878	0.986333	5	2006-11-27
566779	808	815	0.991411	5	2009-12-13
235722	580	593	0.978078	1	2011-07-01
222937	491	569	0.862917	3	2008-05-31

In [33]:
```python
Helpfull.tail()
```

Out[33]:

	HelpfulnessNumerator	HelpfulnessDenominator	Helpfulness	Score	Time
202645	1	1	1.0	2	2008-09-05
490640	1	1	1.0	5	2011-12-09
490639	1	1	1.0	5	2012-05-03
196610	1	1	1.0	5	2011-05-04
196614	1	1	1.0	4	2011-04-12

In [34]:
```python
help_rate=list(set(Helpfullness['Helpfulness']))
print(f"The help_rate (Helpfulness) has {len(help_rate)} unique values")
```

```
The help_rate (Helpfulness) has 952 unique values
```

In [35]:
```python
help_rate=pd.DataFrame(help_rate, columns=['Help_Rate']
            ).sort_values(by=['Help_Rate'],ascending=False).reset_index()

help_rate['Help_Rate']
```

Out[35]:
```
0      3.000000
1      1.500000
2      1.000000
3      0.996894
4      0.996198
         ...
947    0.022222
948    0.021277
949    0.010989
950    0.000000
951   -1.000000
Name: Help_Rate, Length: 952, dtype: float64
```

In [36]:

```
pd.cut(df['Helpfulness'] , bins = [-1, 0, 0.2, 0.4, 0.6, 0.8, 1.0],
       labels = ['Empty', '0-20%', '20-40%', '40-60%', '60-80%', '80-100%'])
```

Out[36]:

```
0          80-100%
1              NaN
2          80-100%
3          80-100%
4              NaN
            ...
568449         NaN
568450         NaN
568451     80-100%
568452     80-100%
568453         NaN
Name: Helpfulness, Length: 568411, dtype: category
Categories (6, object): ['Empty' < '0-20%' < '20-40%' < '40-60%' < '60-80%' < '8
0-100%']
```

In [37]:

```
df['Help_Rate']=pd.cut(df['Helpfulness'] ,
                bins = [-1,        0,         0.2,      0.4,       0.6,      0.8, 1.0]
                labels = ['Empty', '0-20%', '20-40%', '40-60%', '60-80%', '80-100%
df['Help_Rate']
```

Out[37]:

```
0          80-100%
1              NaN
2          80-100%
3          80-100%
4              NaN
            ...
568449         NaN
568450         NaN
568451     80-100%
568452     80-100%
568453         NaN
Name: Help_Rate, Length: 568411, dtype: category
Categories (6, object): ['Empty' < '0-20%' < '20-40%' < '40-60%' < '60-80%' < '8
0-100%']
```

In [38]:

```
df.groupby(['Score','Help_Rate']).agg('count')
```

Out[38]:

		Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
Score	**Help_Rate**						
1	**Empty**	8060	8060	8060	8060	8060	
	0-20%	2338	2338	2338	2338	2338	
	20-40%	4649	4649	4649	4649	4649	
	40-60%	6586	6586	6586	6586	6586	
	60-80%	5836	5836	5836	5836	5836	
	80-100%	12531	12531	12531	12531	12531	
2	**Empty**	4234	4234	4234	4234	4234	
	0-20%	737	737	737	737	737	
	20-40%	1618	1618	1618	1618	1618	
	40-60%	3051	3051	3051	3051	3051	
	60-80%	2486	2486	2486	2486	2486	
	80-100%	7014	7014	7014	7014	7014	
3	**Empty**	5062	5062	5062	5062	5062	
	0-20%	474	474	474	474	474	
	20-40%	1506	1506	1506	1506	1506	
	40-60%	3384	3384	3384	3384	3384	
	60-80%	2754	2754	2754	2754	2754	
	80-100%	11036	11036	11036	11036	11036	
4	**Empty**	4780	4780	4780	4780	4780	
	0-20%	116	116	116	116	116	
	20-40%	909	909	909	909	909	
	40-60%	3185	3185	3185	3185	3185	
	60-80%	2941	2941	2941	2941	2941	
	80-100%	26707	26707	26707	26707	26707	
5	**Empty**	11638	11638	11638	11638	11638	
	0-20%	432	432	432	432	432	
	20-40%	2275	2275	2275	2275	2275	
	40-60%	10312	10312	10312	10312	10312	
	60-80%	11060	11060	11060	11060	11060	
	80-100%	140659	140659	140659	140659	140659	1

In [39]:

```
df_summary=df.groupby(['Score','Help_Rate']).agg({'Id':'count'}).reset_index()
df_summary
```

Out[39]:

	Score	Help_Rate	Id
0	1	Empty	8060
1	1	0-20%	2338
2	1	20-40%	4649
3	1	40-60%	6586
4	1	60-80%	5836
5	1	80-100%	12531
6	2	Empty	4234
7	2	0-20%	737
8	2	20-40%	1618
9	2	40-60%	3051
10	2	60-80%	2486
11	2	80-100%	7014
12	3	Empty	5062
13	3	0-20%	474
14	3	20-40%	1506
15	3	40-60%	3384
16	3	60-80%	2754
17	3	80-100%	11036
18	4	Empty	4780
19	4	0-20%	116
20	4	20-40%	909
21	4	40-60%	3185
22	4	60-80%	2941
23	4	80-100%	26707
24	5	Empty	11638
25	5	0-20%	432
26	5	20-40%	2275
27	5	40-60%	10312
28	5	60-80%	11060
29	5	80-100%	140659

In [40]:

```
# Create Pivot Table for better appreciation

df_summary.pivot(index='Help_Rate',columns='Score')
```

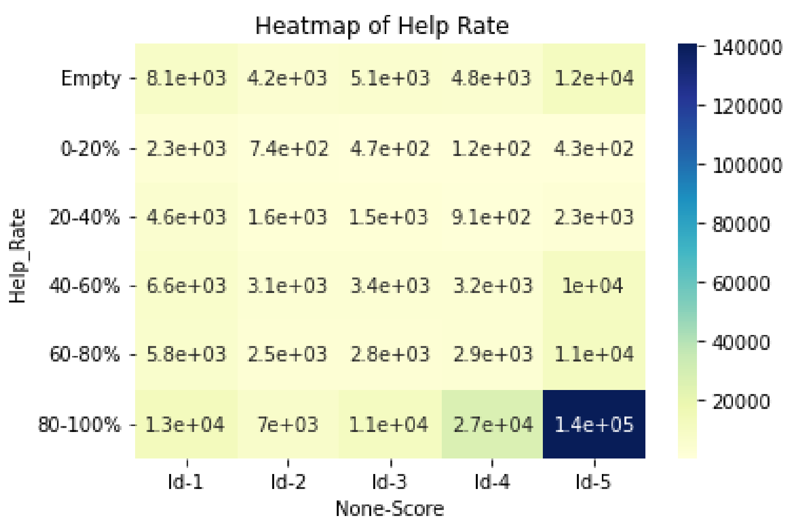
Out[40]:

	Id				
Score	**1**	**2**	**3**	**4**	**5**
Help_Rate					
Empty	8060	4234	5062	4780	11638
0-20%	2338	737	474	116	432
20-40%	4649	1618	1506	909	2275
40-60%	6586	3051	3384	3185	10312
60-80%	5836	2486	2754	2941	11060
80-100%	12531	7014	11036	26707	140659

In [41]:

```
# Create heatmap of it, for better Visualizations

sns.heatmap(df_summary.pivot(index='Help_Rate',columns='Score'),
            annot=True , cmap = 'YlGnBu')
plt.title('Heatmap of Help Rate  ')
plt.show()
```



In [42]:

```
# Key message from above:
# Reviews are skewed towards positive
# Many people had up-vote, score 5 reviews
# More than half of the reviews had zero votes
```

In [43]:

```
score_list=list(set(df['Score'].unique()))
score_list
```

Out[43]: [1, 2, 3, 4, 5]

```
Nore:
The Scores will be converted into two categories, viz:
    1 for Scores of 4 or 5
    0 for Scores of 1 or 2
    Scores of 3 are Neutral scores and will be excluded

==> Score_dict={1:0, 2:0, 4:1, 5:5}
```

In [44]:
```python
# Converting the Scores into 2 categories 0 & 1
# The map() function takes two inputs as a function and an iterable object.
#    The function that is given to map() is a normal function, and it will
#    iterate over all the values present in the iterable object given.

dbf = df[df['Score'] != 3]                          # Exclude Scores of 3
X = dbf['Text']
Score_dict = {1:0, 2:0, 4:1, 5:1}
y = dbf['Score'].map(Score_dict)
```

In [45]:
```python
y
```

Out[45]:
```
0         1
1         0
2         1
3         0
4         1
         ..
568449    1
568450    0
568451    1
568452    1
568453    1
Name: Score, Length: 525773, dtype: int64
```

In [46]:
```python
X.head()
```

Out[46]:
```
0    I have bought several of the Vitality canned d...
1    Product arrived labeled as Jumbo Salted Peanut...
2    This is a confection that has been around a fe...
3    If you are looking for the secret ingredient i...
4    Great taffy at a great price.  There was a wid...
Name: Text, dtype: object
```

In [47]:
```python
# Convert text to vectors using NLP

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(stop_words = 'english')
```

In [48]:
```python
# Vectorize X

X_vectorized = count_vect.fit_transform(X)
```

In [49]:
```python
print(f"Number of features: {format_number(X_vectorized.shape[1])}")
```

```
Number of features: 114,967
```

In [61]:

```
print(X_vectorized[0])
```

```
  (0, 22339)	1
  (0, 110162)	1
  (0, 25060)	1
  (0, 38563)	1
  (0, 46694)	1
  (0, 83025)	1
  (0, 50509)	1
  (0, 84561)	1
  (0, 83005)	2
  (0, 64821)	1
  (0, 63908)	1
  (0, 98504)	1
  (0, 82915)	1
  (0, 67769)	1
  (0, 95333)	1
  (0, 20373)	2
  (0, 62371)	1
  (0, 45571)	1
  (0, 9225)	1
```

Some Helper Functions

In [62]:

```
# ML model Training ---> will be called with different model functions

def model_fit(X, y, NLP_model, ML_model, show_coeff=1):

    X_vectorized = NLP_model.fit_transform(X)
    print(f"Number of features: {format_number(X_vectorized.shape[1])}")

    X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y,
                                            random_state=42, test_size=0.2)

    print(f' Training records(X_train): {format_number(X_train.shape[0])}')
    print(f' Test     records(X_test) : {format_number(X_test.shape[0])}')

    ML = ML_model.fit(X_train, y_train)
    accuracy = ML.score(X_test, y_test)
    print (f'Model Accuracy: {round(accuracy*100,2)}%');

    if show_coeff == 1:
        word  = NLP_model.get_feature_names()
        coeff = ML.coef_.tolist()[0]

        coeff_df = pd.DataFrame(zip(word, coeff),columns=['Word', 'Coefficeient']
                      ).sort_values(['Coefficeient', 'Word'],ascending=False)
        print('\n')
        print("\n===== Top 20 Positives =====")
        print(coeff_df[['Word','Coefficeient']].head(20).to_string(index=False))

        print('\n')
        print("===== Top 20 Negatives =====")
        print(coeff_df[['Word','Coefficeient']].tail(20).to_string(index=False))
```

In [63]:

```
# Prediction Function

def model_predict(X, y, NLP_model, ML_model):

    X_vectorized = NLP_model.fit_transform(X)
    print(f"Number of features: {format_number(X_vectorized.shape[1])}")

    X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y,
                                                        random_state=42, test_size=0.2)
    print(f' Training records(X_train): {format_number(X_train.shape[0])}')
    print(f' Test     records(X_test) : {format_number(X_test.shape[0])}')

    ML = ML_model.fit(X_train, y_train)
    predictions=ML.predict(X_test)
    CM=confusion_matrix(predictions,y_test)    # Confusion Matrix

    print("\nConfusion Matrix:\n",CM,"\n")
    accuracy=accuracy_score(predictions,y_test)
    print (f'Model Accuracy: {round(accuracy*100,2)}%');
    print(f"\nClassification Report:\n {classification_report(y_test, predictions)
```

Step 3: The ML Model

Train -> Test -> Split

In [64]:

```
# Train Test Split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y,
                                                    random_state=42, test_size=0.2)
```

In [65]:

```
print(f' Training records(X_Train): {format_number(X_train.shape[0])}')
```

```
 Training records(X_Train): 420,618
```

Model Training

In [66]:

```
from sklearn.linear_model import LogisticRegression
logistic=LogisticRegression()
```

In [67]:

```
model_fit(X, y, count_vect, LogisticRegression())
```

```
Number of features: 114,967
 Training records(X_train): 420,618
 Test     records(X_test) : 105,155
Model Accuracy: 93.73%



===== Top 20 Positives =====
       Word  Coefficeient
 pleasantly      3.933869
   downside      3.494289
  addicting      3.263224
  skeptical      2.781742
     delish      2.444362
     hooked      2.419159
     resist      2.292474
   tastiest      2.256386
   drawback      2.239924
     solved      2.220911
    easiest      2.218475
    settled      2.205177
        met      2.180962
    worries      2.161353
   hesitant      2.156793
      saves      2.121418
  excellent      2.117856
 economical      2.117643
       whim      2.088638
exceptional      2.062045


===== Top 20 Negatives =====
          Word  Coefficeient
 dissapointing     -2.355627
         blech     -2.376577
     redeeming     -2.503894
         ruins     -2.540074
       defeats     -2.550337
       vomited     -2.566772
disappointment     -2.569603
   disapointed     -2.689423
        lacked     -2.779702
        ripoff     -2.875893
     cancelled     -2.941167
   embarrassed     -2.965753
      mediocre     -2.985264
         lousy     -3.037387
 disappointing     -3.081501
    returnable     -3.173118
         worst     -3.376063
  unacceptable     -3.520919
   undrinkable     -3.619085
     deceptive     -3.762140
```

Predictions

In [68]:
```
# Load the necessary modules for prediction

from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [69]:
```
# Get predictions with LogisticRegression model

model_predict(X,y,count_vect, logistic)
```

```
Number of features: 114,967
 Training records(X_train): 420,618
 Test     records(X_test) : 105,155

Confusion Matrix:
 [[12123  2370]
 [ 4223 86439]]

Model Accuracy: 93.73%

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.74      0.79     16346
           1       0.95      0.97      0.96     88809

    accuracy                           0.94    105155
   macro avg       0.89      0.86      0.87    105155
weighted avg       0.94      0.94      0.94    105155
```

In [70]:
```
# Accuracy is around 93.7% - not bad.
# However we notice that some of those significant coefficients are not meaningful
# Use dummy classifier to rectify the anamaly

from sklearn.dummy import DummyClassifier
```

In [71]:
```
# Train the model with DummyClassifier

model_fit(X, y, count_vect, DummyClassifier(), 0)
```

```
Number of features: 114,967
 Training records(X_train): 420,618
 Test     records(X_test) : 105,155
Model Accuracy: 84.46%
```

In [72]:

```python
# Term Frequency – Inverse Document Frequency (TF-IDF) model
# Logistic regression model on TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf = TfidfVectorizer(stop_words = 'english')
model_fit(X, y, tf_idf, LogisticRegression())
```

```
Number of features: 114,967
 Training records(X_train): 420,618
 Test     records(X_test) : 105,155
Model Accuracy: 93.54%



===== Top 20 Positives =====
      Word  Coefficeient
     great     13.349196
 delicious     12.554462
      best     12.030686
   perfect     10.483679
 excellent      9.772532
     loves      9.677597
    highly      9.045936
      love      8.614605
 wonderful      7.971477
    hooked      7.903948
   amazing      7.868160
   awesome      7.637125
  favorite      7.472283
      good      7.432965
      nice      7.401860
pleasantly      7.187297
     yummy      6.753718
 fantastic      6.710821
    smooth      6.590177
   pleased      6.576301


===== Top 20 Negatives =====
          Word  Coefficeient
         sadly     -5.846292
       sounded     -6.062681
         worse     -6.111581
    disgusting     -6.154328
   undrinkable     -6.232803
         bland     -6.373408
         stale     -6.449472
          yuck     -6.457425
        return     -6.518027
     tasteless     -6.570037
         threw     -6.829738
          weak     -7.122722
 unfortunately     -7.545472
      horrible     -7.988499
   disappointed     -8.388755
disappointment     -8.509028
         awful     -8.920553
      terrible     -9.406202
```

```
disappointing     -9.420447
        worst    -11.784033
```

In [73]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf = TfidfVectorizer(stop_words = 'english')
model_predict(X, y, tf_idf, LogisticRegression())
```

```
Number of features: 114,967
 Training records(X_train): 420,618
 Test     records(X_test) : 105,155

Confusion Matrix:
 [[11437  1887]
 [ 4909 86922]]

Model Accuracy: 93.54%

Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.70      0.77     16346
           1       0.95      0.98      0.96     88809

    accuracy                           0.94    105155
   macro avg       0.90      0.84      0.87    105155
weighted avg       0.93      0.94      0.93    105155
```

Accuracy roughly the same, 93.6%. However, the significant words make much more sense now, with higher coefficient magnitude as well!

In [74]:

```python
# Help_Rate Prediction --> since the score vote of 3 are neutral reviews,
#   we will get rid of all votes with score of 3

df_main = df[df['Score'] == 5]
df_main.shape
```

Out[74]: (363111, 12)

In [75]:

```python
data = df_main[df_main['Help_Rate'].isin(['0-20%', '20-40%', '60-80%', '80-100%'])
data.shape
```

Out[75]: (154426, 12)

In [76]:

```python
X = data['Text']
X.head()
```

Out[76]:

```
0     I have bought several of the Vitality canned d...
8     Right now I'm mostly just sprouting this so my...
10    I don't know if it's the cactus or the tequila...
11    One of my boys needed to lose some weight and ...
14    The Strawberry Twizzlers are my guilty pleasur...
Name: Text, dtype: object
```

In [77]:
```
score_dict = {'0-20%': 0, '20-40%': 0, '60-80%': 1, '80-100%': 1}
y = data['Help_Rate'].map(score_dict)
print(y)
```

```
0         1.0
8         1.0
10        1.0
11        1.0
14        1.0
         ...
568440    1.0
568444    1.0
568445    1.0
568451    1.0
568452    1.0
Name: Help_Rate, Length: 154426, dtype: float64
```

In [78]:
```
y_values=y.value_counts()
pd.DataFrame(y_values.values, index=y_values.index,
             columns=['Score:------->Total _Count'])
```

Out[78]:

	Score:------->Total _Count
1.0	151719
0.0	2707

```
!pip install imbalanced-learn
```

In [79]:
```
# The target class 'y' is highly skewed , we will observe that positive
#    upvotes(151719, 98.2%) are too much higher than negative
#    votes(2707, 1.8%) there is therefore need to resample the data for
#    improved balance:

from sklearn.feature_extraction.text import TfidfVectorizer
```

In [80]:
```
TF_Id = TfidfVectorizer()
```

In [81]:
```
X_tf = TF_Id.fit_transform(X)
```

In [82]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_tf,y,
                                                test_size=0.2,random_state=3)
```

In [83]:
```
y_test.value_counts()
```

Out[83]:
```
1.0    30344
0.0      542
Name: Help_Rate, dtype: int64
```

In [84]:
```
## RandomOverSampler to handle imbalanced data

#import imblearn
from imblearn.over_sampling import RandomOverSampler
```

In [85]:
```
r_o_s =  RandomOverSampler()
```

In [86]:
```
X_train_resampled, y_train_resampled = r_o_s.fit_resample(X_tf, y)
```

In [87]:
```
print(f"X_train(resampled)   {X_train_resampled.shape}   |   ", end=" ")
print(f"y_train(resampled) {y_train_resampled.shape}")
```

```
X_train(resampled)   (303438, 67507)   |   y_train(resampled) (303438,)
```

In [88]:
```
from collections import Counter
```

In [89]:
```
from sklearn.linear_model import LogisticRegression
log_class=LogisticRegression()

#X_train = X_train_resampled
#y_train = y_train_resampled

clf=log_class.fit(X_train_resampled, y_train_resampled)
```

In [90]:
```
y.value_counts()
```

Out[90]:
```
1.0    151719
0.0      2707
Name: Help_Rate, dtype: int64
```

In [91]:
```
# Original X_train values before Oversampling

y_values=y_train.value_counts()
pd.DataFrame(y_values.values, index=y_values.index,
             columns=['Score:------->y_train: Total _Count'])
```

Out[91]:

	Score:------->y_train: Total _Count
1.0	121375
0.0	2165

In [92]:
```
# Original X_train values after Oversampling

y_values=y_train_resampled.value_counts()
pd.DataFrame(y_values.values, index=y_values.index,
             columns=['Score:------->y_train: Total _Count'])
```

Out[92]:

	Score:------->y_train: Total _Count
1.0	151719
0.0	151719

In [93]:
```
print(f'Original  dataset shape {Counter(y_train)}')
print(f'Resampled dataset shape {Counter(y_train_resampled)}')
```

```
Original  dataset shape Counter({1.0: 121375, 0.0: 2165})
Resampled dataset shape Counter({1.0: 151719, 0.0: 151719})
```

To improve our model, We will employ GridSearchCV which is a process of performing hyperparameter tuning in order to determine the optimal values for a given model. GridSearchCV loops through predefined hyperparameters and fit the estimator (model) on the training set. So, that in the end best parameters from the listed hyperparameters is selected.

In [94]:

```
from sklearn.model_selection import GridSearchCV
```

In [95]:

```
# define a set of grid parameters to iterate

param_grid={'C':10.0 **np.arange(-2,3),'penalty':['l1','l2']}

# GridSearch with LogisticRegression --> log_class
grid = GridSearchCV(log_class, param_grid, refit = True, verbose = 3,n_jobs=-1)
```

In [96]:

```
# Train the model with the new hyper-parameters and predict
#     Predict
#     Find the accuracy
#     Show classification report

grid.fit(X_train_resampled, y_train_resampled)
grid_predictions = grid.predict(X_test)
accuracy         = accuracy_score(y_test, grid_predictions)

# print best parameter after tuning
print(grid.best_params_)

print(f"\nConfusion Matrix:\n {confusion_matrix(y_test,grid_predictions)}")
print (f'\nModel Accuracy: {round(accuracy*100,2)}%');
print()

# print classification report
print(f"Classification Report:\n  {classification_report(y_test, grid_predictions)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'C': 100.0, 'penalty': 'l2'}

Confusion Matrix:
 [[  542     0]
 [  433 29911]]

Model Accuracy: 98.6%

Classification Report:
               precision    recall  f1-score   support

         0.0       0.56      1.00      0.71       542
         1.0       1.00      0.99      0.99     30344

    accuracy                           0.99     30886
   macro avg       0.78      0.99      0.85     30886
weighted avg       0.99      0.99      0.99     30886
```

In [97]:

```
# Fantastically more balance results and predictions with optimal
#                                  hyper-parameters of {'C':100}
```

```
recall  = TP / (TP + FN)

The recall is the measure of our model correctly identifying True Positives.
Thus, for all the customers who actually had a vote of 5, recall tells us how many
we correctly identified as customers who had an up-vote (score=5). in this instance,
99% of customers who had an up-vote were correctly identified

precision of class 0 = TP of class 0/total number of object
The precision of our model is 100%! which implies that when it predicts that a
customer has up-vote(4 or 5) it is most of the time.
            precision of class 1 = TP of class 1/total number of object

            macro average = (precision of class 0 + precision of class 1)/2

weighted average is precision of all classes merge together
weighted average = (TP of class 0 + TP of class 1)/(total number of class 0 + total
number of class 1)

TP   FN        [  542      0]
FP   TN        [  433  29911]

f1-score is a measure of a model's accuracy on a dataset
a good F1 score means that you have low false positives and low false negatives,
Accuracy is used when the True Positives and True negatives are more important while
f1-score is used when the False Negatives and False Positives are crucial.

Support is the number of actual occurrences of the class in the specified dataset.
```

In [ ]: