TEXT MINING PROJECT: NATURAL LANGUAGE PROCESSING MODELS TO PREDICT CUSTOMER SENTIMENTS FROM AMAZON FINE FOOD REVIEW

HTTPS://GITHUB.COM/YAY66/NLP-AMAZON-REVIEW/BLOB/MAIN/TEXT_MINING_PROJECT_NLP_AMAZON_REVIEWS.IPYNB

ISM 6359 Data Mining

MARCH 11, 2023

YAYUAN ZHANG

Contents

Abstract

The food industry is one of the most important industries in the global economy, which is also an industry with relatively strong competition. As one of the largest online retailers of food, Amazon must gain a deep understanding of customer preferences and sentiments to make informed decisions about its products.

This project focuses on developing natural language processing models to predict customer sentiments from Amazon Fine Food Reviews. Also, the project aims to help businesses gain insights into customer preferences and sentiments to make informed decisions. In the project, various data preprocessing techniques are detailedly described to prepare the data for analysis.

Additionally, various natural language processing techniques are employed to learn the relationships between the text data and the corresponding sentiment labels. The stakeholders can understand the strengths and weaknesses of their products and develop strategies and plan to improve their products' quality and customer satisfaction.

Keywords: Text Mining, Data Pre-Processing, Sentiment analysis, Amazon Fine Food Review, Natural Language Processing, Machine Learning.

1. Background

1.1 State Text Mining Tool

In this process, the Python programming language that acts on the PyCharm and Jupyter platforms is my text mining tool and environment.

1.2 Stakeholders

- Food industries: the food industries that sell their product on Amazon. Their businesses rely heavily on customer feedback to improve their products and increase customer satisfaction. Therefore, this project can benefit businesses in the food industry by providing insights into customer preferences and sentiment by predicting customer sentiment.
- Consumers: The consumers who purchase food products on Amazon also can benefit from this project. Businesses can improve product quality and increase customer satisfaction by analyzing customer sentiments, resulting in a better shopping experience for consumers.

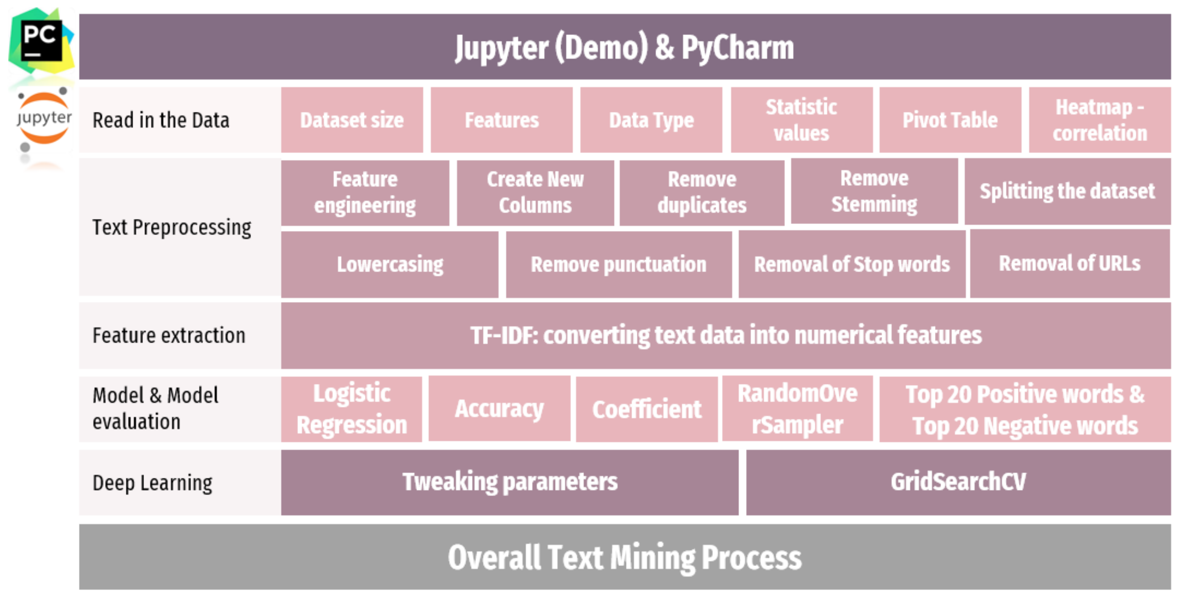
1.3 Data & Source

The dataset comes from Kaggle. The reviews were collected from October 1999 to October 2012. The dataset contains 568454 data points and 10 features. Each data point represents each customer reviews for various food products sold on Amazon, including the Product ID, Helpfulness Numerator, Helpfulness Denominator, Score, Summary, review text, etc.

The dataset is a fantastic resource for assessing consumer sentiments since it offers a thorough collection of client comments. The dataset's breadth and diversity also make it appropriate for training NLP models, facilitating the creation of precise sentiment analysis algorithms.

2. Data Analysis

2.1 Workflow Analysis process



2.2 Import libraries

First, I import the required libraries, including the science calculation, parts mainly for visualization, packages for pre-processing text data, model algorithms, ignoring some warnings, and so on.

2.3 Data Understanding

2.3.1 Basic information

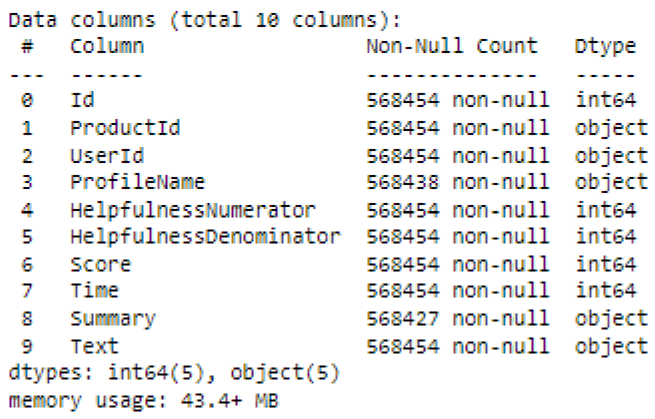
The data has 568454 observations with 10 features. The target variable is the score. Other features are named: 'Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Time', 'Summary', and 'Text'.

```
# Checking the head of data and defining the target variable
df.head()
```

- # Checking the data frame size to learn what amount of data are dealing with
- df.shape

In the dataset, 5 features are numeric fields, other 5 features are object fields.

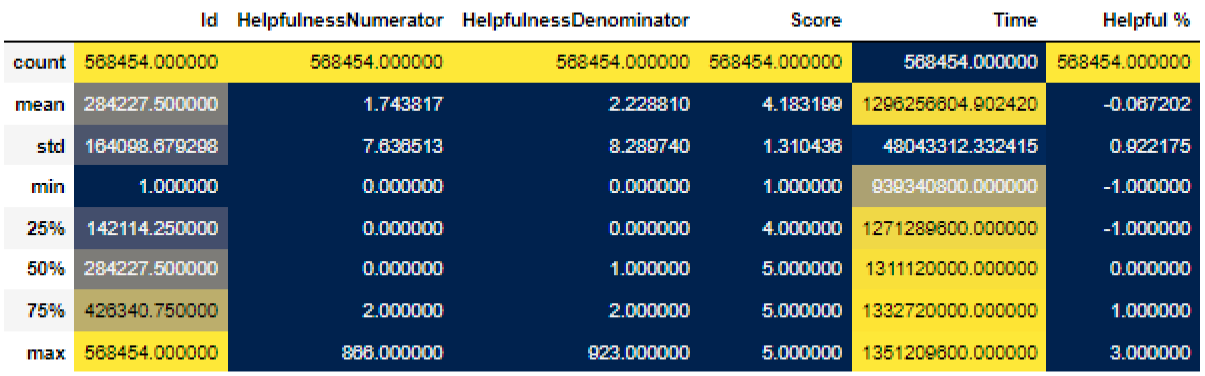
- # Checking the datatypes for all features/columns & whether have empty cell or not
- df.info()



```
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype 
---  ------                  --------------   ----- 
 0   Id                      568454 non-null  int64 
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568438 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64 
 5   HelpfulnessDenominator  568454 non-null  int64 
 6   Score                   568454 non-null  int64 
 7   Time                    568454 non-null  int64 
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

Next, I checked the distribution of the values of all the numeric fields. the describe function checks the distribution of the values of all the numeric fields, so it gets the counts, means, standard deviations, etc. for each feature.

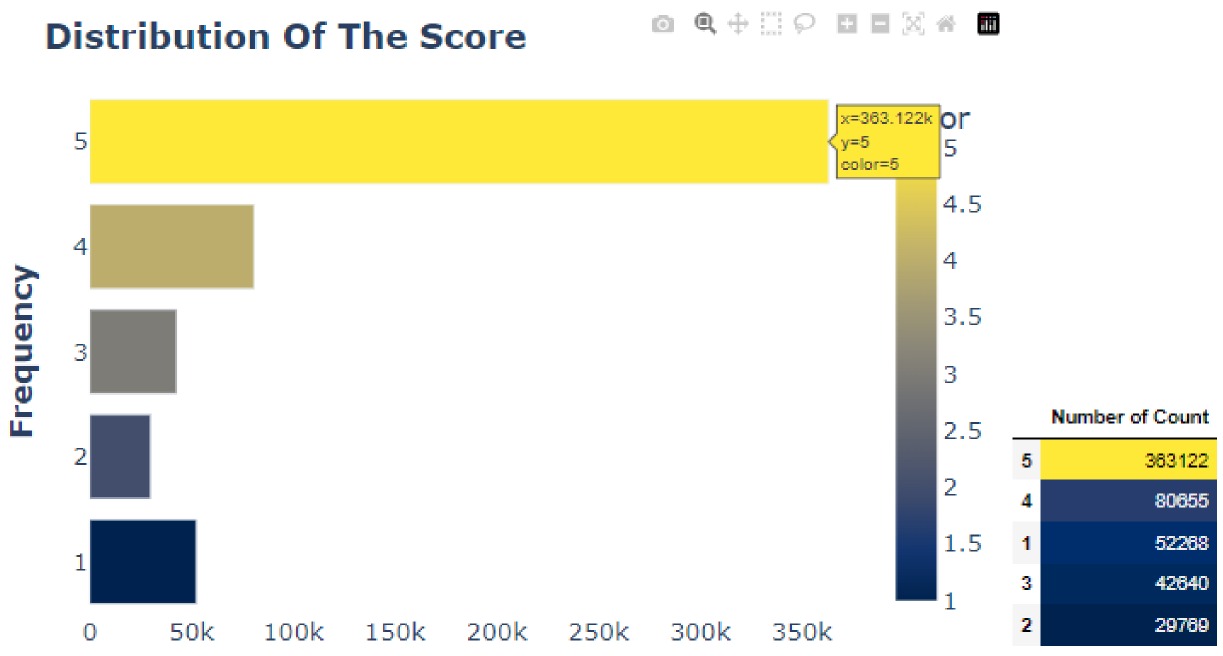
- # Reading the data error description to check the spread of values across all numerical fields
- data.describe().style.background_gradient(cmap='cividis')



	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Helpful %
count	568454.000000	568454.000000	568454.000000	568454.000000	568454.000000	568454.000000
mean	284227.500000	1.743817	2.228810	4.183199	1296256804.902420	-0.067202
std	164098.679298	7.636513	8.289740	1.310436	48043312.332415	0.922175
min	1.000000	0.000000	0.000000	1.000000	939340800.000000	-1.000000
25%	142114.250000	0.000000	0.000000	4.000000	1271289600.000000	-1.000000
50%	284227.500000	0.000000	1.000000	5.000000	1311120000.000000	0.000000
75%	426340.750000	2.000000	2.000000	5.000000	1332720000.000000	1.000000
max	568454.000000	866.000000	923.000000	5.000000	1351209600.000000	3.000000

2.3.2 EDA

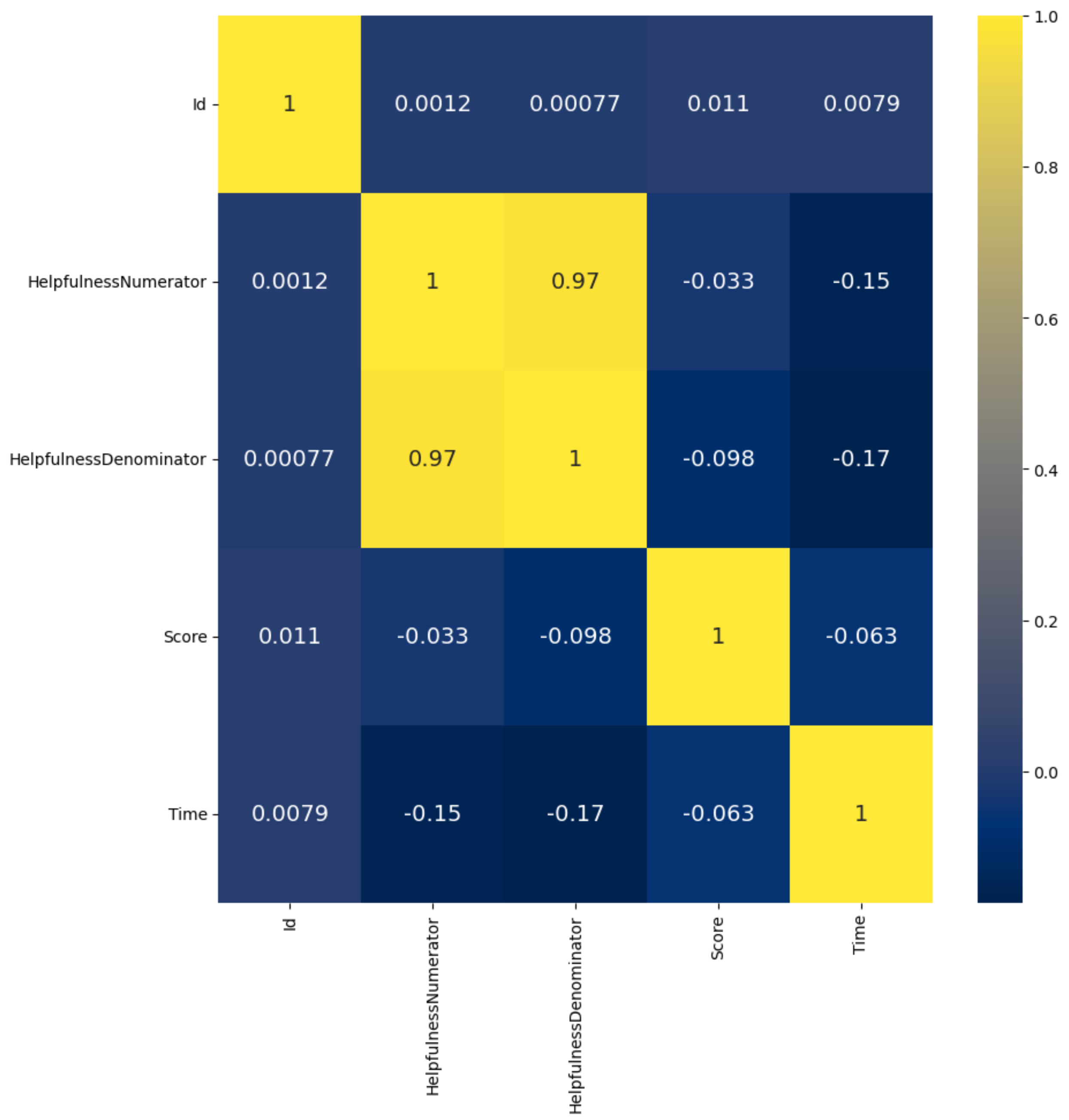
Before any feature engineering, I prefer to quickly view the original data distribution by using data visualization to gain more insight and understand the relationship between each feature.



	Number of Count
5	363122
4	80655
1	52268
3	42640
2	29769

According to the score distribution bar chart above, a large number of reviews are 5 points, a total of 363122.

- vc = df['Score'].value_counts().to_frame().reset_index()
- fig = px.bar(x=vc["Score"], y=vc["index"], orientation='h',color=vc['index'],labels=True,color_continuous_scale="cividis")
- fig.update_layout(title = "<b> Distribution Of The Score </b>",xaxis_title="",yaxis_title="<b> Frequency </b> ",font_size=18, plot_bgcolor="#fff",showlegend=False)



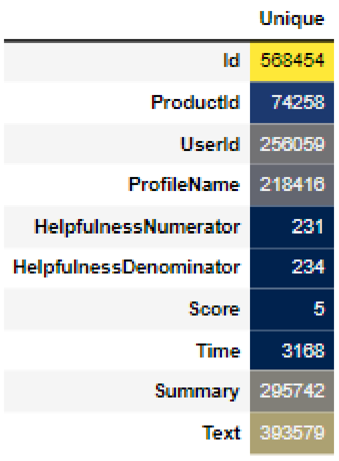
Normally, if we fit highly correlated data in the model, it results in an overfitting problem. The good news is there is no highly correlated feature with the score in this data set.

- plt.figure(figsize=(10, 10))
- sns.heatmap(df.corr(), annot=True, cmap="cividis", annot_kws={"size":14})

2.3.3 Checking total unique values in each column.

I used the following coding to check the total unique values in each column.

```
dict={}
for i in df.columns:
    dict[i]=df[i].value_counts().shape[0]
pd.DataFrame(dict,index=['Unique']).T.style.background_gradient(cmap='cividis')
```



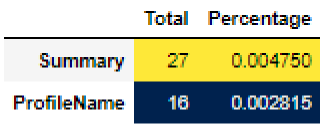
	Unique
Id	568454
ProductId	74258
UserId	256059
ProfileName	218416
HelpfulnessNumerator	231
HelpfulnessDenominator	234
Score	5
Time	3168
Summary	295742
Text	393579

2.3.4 Checking Missing values.

After checking the missing values and the percentage of missing values in the column by the following codes. Two features have missing values, Summary, and Profile Name. It seems like a small percentage. In this case, I made some optimizations to the code of past data mining projects, so that the number of missing values and its percentage can be directly displayed.

```
def missing_data(df):
    total = df.isnull().sum().sort_values(ascending = False)
    Percentage = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending = False)
    return pd.concat([total, Percentage], axis=1, keys=['Total', 'Percentage'])
```

```
missing_data(df)[missing_data(df)['Percentage']!=0].style.background_gradient(cmap='cividis')
```



	Total	Percentage
Summary	27	0.004750
ProfileName	16	0.002815

2.4 Data Pre-processing

Next, I will start doing data preprocessing to help improve the accuracy of the model.

2.4.1 Adding new column for helpful%

From the original dataset above I can see that there is a numerator and a denominator, so I will create a new column called "Helpful %" to add to the DataFrame (df for short) and calculate a value for each row representing that review "Useful ratio". Use the formula: helpfulnessnumerator/helpfulnessdenominato. If the comment has not been rated by anyone, the value of this column is -1.

```
df['Helpful %'] = np.where(df['HelpfulnessDenominator'] > 0, df['HelpfulnessNumerator'] /
df['HelpfulnessDenominator'], -1)
```

2.4.2 Adding new column for upvote metrics.

Then, I cut the data into slices and make them into 7 bins, divide the column into 6 levels according to the percentage type as ['Empty' < '0-20%' < '20-40%' < '40-60%' < '60-80%' < '80-100 %'].

```
df['%upvote'] = pd.cut( df['Helpful %'] , 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.head()
```

2.4.3 Assigning different labels

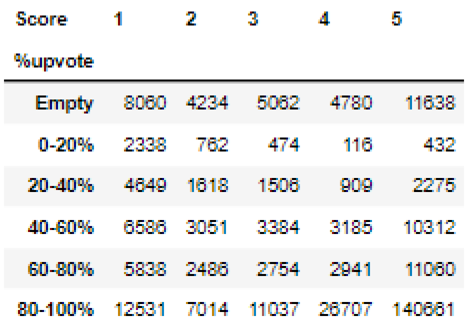
I assigned different labels to helpful% based on %upvote to its value, which counts data based on the score as a category and %upvote as a subcategory. In this step, I organized the table that shows only the ID column and reset the index.

- df.groupby(['Score', '%upvote']).agg({'Id':'count'})
- df_s2=df.groupby(['Score', '%upvote']).agg({'Id':'count'}).reset_index()

2.4.4 Creating Pivot Table and visualization for above table

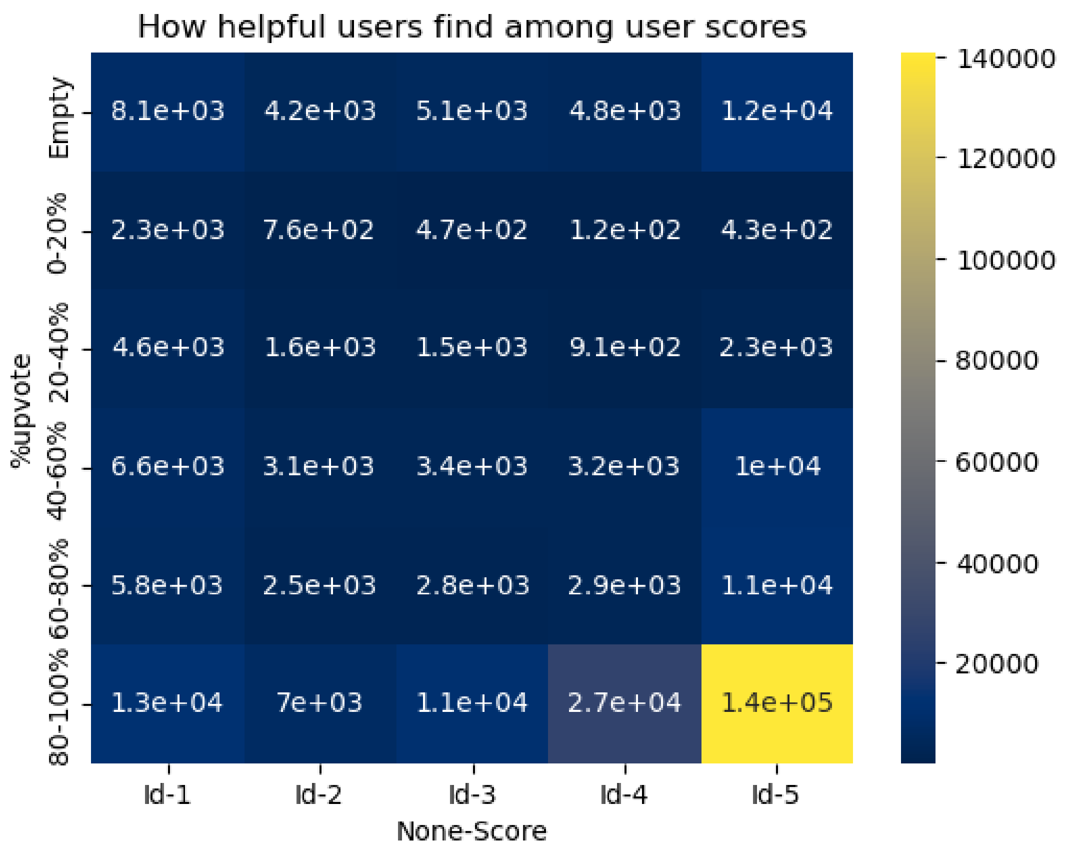
In this step, I created a PivotTable for better conclusions and a heatmap for it.

- df_s2.pivot(index='%upvote',columns='Score')



Score	1	2	3	4	5
%upvote					
Empty	8060	4234	5062	4780	11638
0-20%	2338	762	474	116	432
20-40%	4649	1618	1506	909	2275
40-60%	6586	3051	3384	3185	10312
60-80%	5838	2486	2754	2941	11060
80-100%	12531	7014	11037	26707	140661

- sns.heatmap(df_s2.pivot(index='%upvote',columns='Score'),annot=True,cmap = 'cividis')
- plt.title('How helpful users find among user scores')



Based on the table above, it is seeing the key information:

- ☐ Reviews are skewed toward positive.
- ☐ More than half of the reviews are with zero votes.
- ☐ Lots of people rank 5-points for the review.

2.5 NLP – Converting text into vector & Data cleaning

Converting text into the vector is a key step in the NLP text mining project because computers cannot directly process text data. By converting text into vectors, we can use various machine learning algorithms to train models, classify, cluster, generate text, and many other tasks. Also, data often contains some useless or redundant information, which interferes with the accuracy and efficacy of machine learning algorithms.

2.5.1 Removing the duplicates.

Regarding the data cleaning part, firstly I remove the duplicate data point if any by using the following coding.

```
df.duplicated().sum()

df.drop_duplicates(inplace=True)
```

2.5.2 Lower casting

Then, I convert all text to lowercase this avoids duplication or mistakes due to different cases.

```
df['Text'] = df['Text'].str.lower()
```

2.5.3 Removing punctuation

The text might contain extraneous characters such as punctuation marks, special symbols, numbers, etc. which need to be removed as they are not helpful for analysis. I remove them from the dataset by using the following coding, such as !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.

```
PUNCT_TO_REMOVE = string.punctuation

def remove_punctuation(text):
        """custom function to remove the punctuation"""
        return text.translate(str.maketrans('', '', PUNCT_TO_REMOVE))

df['Text'] = df['Text'].apply(lambda text: remove_punctuation(text))
```

2.5.4 Removing of Stop-words

In this step, I remove stop words in the text, such as "the", "a", "an", etc. These words are not helpful for text mining and thus should be removed.

```
STOPWORDS = set(stopwords.words('english'))
def remove_stopwords(text):
        """custom function to remove the stopwords"""
        return " ".join([word for word in str(text).split() if word not in STOPWORDS])
df['Text'] = df['Text'].apply(lambda text: remove_stopwords(text))
```

2.5.5 Removing URLs

In this step I strip out HTML tags or URLs, as they have no useful information for analysis.

```
def remove_urls(text):
        url_pattern = re.compile(r'https?://\S+|www\.\S+')
        return url_pattern.sub(r'', text)
df['Text'] = df['Text'].apply(lambda text: remove_urls(text))
```

2.6 Checking dataset before next step

```
df.shape
```

```
out: (568454, 12)
```

```
df.head()
```

Before I build the model, I prefer to check dataset again and ensure that all data are ready to use. Now, all data are cleaned and ready to run into next step.

2.7 Binary classification label

The dataset still contains some textual reviews and their ratings, which is 1 to 5 score with 5 being the best. My goal is to categorize these reviews into positive (4 and 5 score) and negative (1 and 2 score) and remove reviews with a rating 3 because it is unclear whether they are positive or negative. Therefore, I use y_dict to convert 1 and 2 in the Score column to 0, and 4 and 5 to 1, indicating that the ratings are divided into two categories, 1 and 2 are negative ratings, and 4 and 5 are positive ratings. Finally, save the transformed Score column into the variable y.

- df['Score'].unique()
- df2 = df[df['Score'] != 3]
- y_dict = {1:0, 2:0, 4:1, 5:1}
- X = df2['Text']
- y = df2['Score'].map(y_dict)

3. Build Model

3.1 Fitting on train & test set

In previous step (2.4.6), I performed a train-test split on the transformed feature vector dataset. So the data are all ready to fit the model.

I chose Logistics Regression as my predictive model since it is commonly used for binary and multi-class classification tasks. Firstly, I initialized the LogisticRegression() class and trained the model using the training dataset.

- log=LogisticRegression()

Then, I evaluated the performance of the trained model on the test dataset, computed the accuracy, and printed it out. Before adjusting the parameters.

- X_train, X_test, y_train, y_test = train_test_split(X_c, y)
- print(' train records: {}'.format(X_train.shape[0]))
- ml =log.fit(X_train, y_train)
- acc = ml.score(X_test, y_test)
- print ('Model Accuracy: {}'.format(acc))

I obtained an accuracy of 93.90%, which seems impressive, but there is still room for improvement.

3.2 Fetching Top 20 Positive & Top 20 Negative words

In this step, I output the coefficients of each feature in the trained logistic regression model to find the ranking of the contribution to sentiment classification. Specifically, I used the get_feature_names() method in the CountVectorizer class to obtain a list of feature names (i.e., words) in the feature vector.

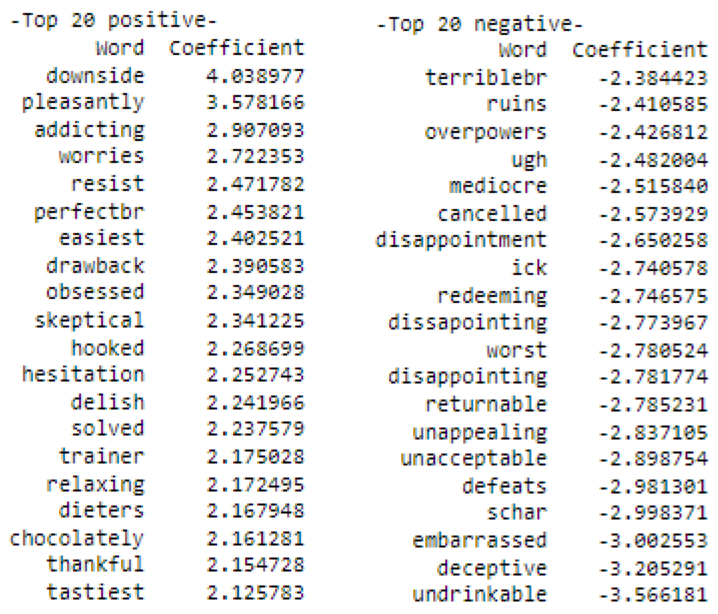
- w = c.get_feature_names()

Then, I obtained the coefficients of each feature from the trained logistic regression model and converted them into a list format. Using the pandas.DataFrame class, I converted the dictionary of feature names and coefficients into a data frame for sorting and outputting.

- coef = ml.coef_.tolist()[0]
- coeff_df = pd.DataFrame({'Word' : w, 'Coefficient' : coef})

Finally, I sorted the data frame in descending order based on the coefficients and outputted the top 20 positive and negative words with the largest and smallest coefficients, respectively. The former are typically words related to positive sentiment, while the latter are typically words related to negative sentiment.

- coeff_df = coeff_df.sort_values(['Coefficient', 'Word'],ascending=False)
- print('-Top 20 positive-')
- print(coeff_df.head(20).to_string(index=False))
- print('\n')
- print('-Top 20 negative-')
- print(coeff_df.tail(20).to_string(index=False))



```
-Top 20 positive-                  -Top 20 negative-
          Word  Coefficient                   Word  Coefficient
      downside     4.038977            terriblebr    -2.384423
    pleasantly     3.578166                 ruins    -2.410585
     addicting     2.907093            overpowers    -2.426812
       worries     2.722353                   ugh    -2.482004
        resist     2.471782              mediocre    -2.515840
     perfectbr     2.453821             cancelled    -2.573929
       easiest     2.402521        disappointment    -2.650258
      drawback     2.390583                   ick    -2.740578
      obsessed     2.349028             redeeming    -2.746575
     skeptical     2.341225          dissapointing    -2.773967
        hooked     2.268699                 worst    -2.780524
    hesitation     2.252743          disappointing    -2.781774
        delish     2.241966            returnable    -2.785231
        solved     2.237579           unappealing    -2.837105
       trainer     2.175028          unacceptable    -2.898754
      relaxing     2.172495               defeats    -2.981301
       dieters     2.167948                 schar    -2.998371
   chocolately     2.161281           embarrassed    -3.002553
      thankful     2.154728             deceptive    -3.205291
       tastiest     2.125783            undrinkable    -3.566181
```

3.3 Evaluate models by applying multiple NLP techniques & multiple ML algorithms

3.3.1 Text classifier function

- def text_fit(X, y, nlp_model,ml_model,coef_show=1):
    - X_c = nlp_model.fit_transform(X)
    - print('features: {}'.format(X_c.shape[1]))

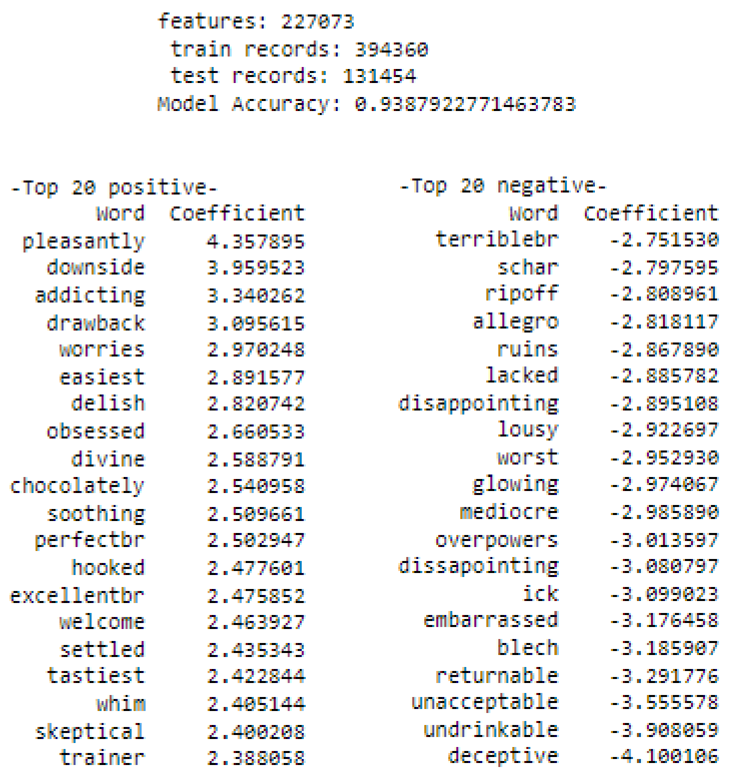
```
X_train, X_test, y_train, y_test = train_test_split(X_c, y)
print(' train records: {}'.format(X_train.shape[0]))
print(' test records: {}'.format(X_test.shape[0]))
ml =ml_model.fit(X_train, y_train)
acc = ml.score(X_test, y_test)
print ('Model Accuracy: {}'.format(acc))
if coef_show == 1:
    w = nlp_model.get_feature_names()
    coef = ml.coef_.tolist()[0]
    coeff_df = pd.DataFrame({'Word' : w, 'Coefficient' : coef})
    coeff_df = coeff_df.sort_values(['Coefficient', 'Word'], ascending=[0, 1])
    print('\n')
    print('-Top 20 positive-')
    print(coeff_df.head(20).to_string(index=False))
    print('\n')
    print('-Top 20 negative-')
    print(coeff_df.tail(20).to_string(index=False))
```

In this step, I create a function for implementing the training and evaluation of a text classifier.

This function is –

- Convert the raw text data X to a numerical feature vector using the fit_transform() method of the incoming nlp_model object;
- Divide the data set into training set and test set, using the train_test_split() method in sklearn;
- Use the passed in ml_model object to fit the training set to get the classifier model ml;
- Predict the test set and calculate the model accuracy acc, using the ml.score() method;

- ☐ If the parameter coef_show is 1, the top 20 positive and negative feature coefficients (Coefficient) after model training are output, that is, the words that have the greatest impact on the classification results, and the output is a pandas data frame.



```
features: 227073
 train records: 394360
 test records: 131454
Model Accuracy: 0.9387922771463783


-Top 20 positive-                -Top 20 negative-
          Word  Coefficient                 Word  Coefficient
    pleasantly     4.357895           terriblebr    -2.751530
      downside     3.959523                schar    -2.797595
     addicting     3.340262               ripoff    -2.808961
      drawback     3.095615              allegro    -2.818117
       worries     2.970248                ruins    -2.867890
       easiest     2.891577               lacked    -2.885782
        delish     2.820742        disappointing    -2.895108
      obsessed     2.660533                lousy    -2.922697
        divine     2.588791                worst    -2.952930
    chocolately    2.540958              glowing    -2.974067
      soothing     2.509661             mediocre    -2.985890
     perfectbr     2.502947           overpowers    -3.013597
        hooked     2.477601        dissapointing    -3.080797
    excellentbr    2.475852                  ick    -3.099023
       welcome     2.463927          embarrassed    -3.176458
       settled     2.435343                blech    -3.185907
      tastiest     2.422844           returnable    -3.291776
          whim     2.405144         unacceptable    -3.555578
     skeptical     2.400208          undrinkable    -3.908059
       trainer     2.388058            deceptive    -4.100106
```

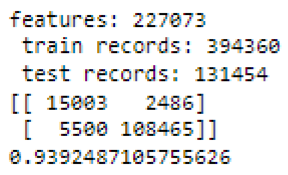
Next, I fit the model using the above function. After refinement, I got 93.88% accuracy.

3.3.2 Prediction function & Confusion Matrix

In this case, I create a prediction function to receive the two models for 'nlp_model' and 'ml_model'. In this case, the function applies a specified NLP model nlp_model to transform the text into features and then fits a specified machine learning model ml_model to predict y based on the features. The function also performs train/test splitting, prints out information about the features and records, computes and prints the confusion matrix and accuracy score of the predictions.

```
def predict(X, y, nlp_model,ml_model):
    X_c = nlp_model.fit_transform(X)
```

- print('features: {}'.format(X_c.shape[1]))
- X_train, X_test, y_train, y_test = train_test_split(X_c, y)
- print(' train records: {}'.format(X_train.shape[0]))
- print(' test records: {}'.format(X_test.shape[0]))
- ml =ml_model.fit(X_train, y_train)
- predictions=ml.predict(X_test)
- cm=confusion_matrix(predictions,y_test)
- print(cm)
- acc=accuracy_score(predictions,y_test)
- print(acc)



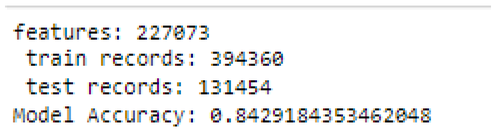
```
features: 227073
 train records: 394360
 test records: 131454
[[ 15003   2486]
 [  5500 108465]]
0.9392487105755626
```

After I fit the model using the above function, I got 93.92% accuracy. It is not bad. However, I noticed that some significant coefficients are not meaningful.

3.3.3 Dummy classifier

The reason to use this classifier is that it is to establish a baseline or benchmark for the model's performance, also provides a reference point to compare the actual model's accuracy and decide whether or not the model is doing better than random guessing.

- text_fit(X, y, c, DummyClassifier(),0)



```
features: 227073
 train records: 394360
 test records: 131454
Model Accuracy: 0.8429184353462048
```

The good news is that I got 84.29% accuracy for this classifier and lower than other twos. Because if the actual learned model does not significantly outperform random guesses, the model needs to be re-evaluated and tuned.

3.3.4 Summary

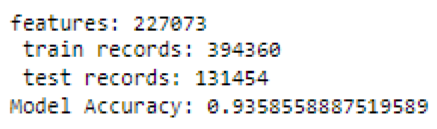
The third function makes use of DummyClassifier, which is essentially a straightforward random classifier that makes results-only estimates rather than using a genuine learning process. Therefore, the low accuracy of the third function may be due to the result of DummyClassifier.

In contrast, the first and second functions use more complex learning algorithms, so they may be more accurate. In addition, the first function outputs feature coefficients, while the second function outputs stricter evaluation criteria such as confusion matrix and accuracy scores, so these functions provide a more comprehensive model evaluation.

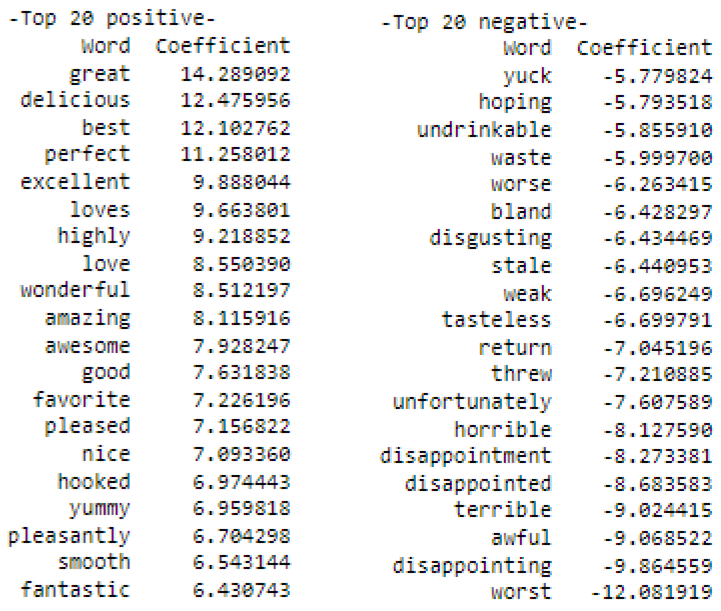
3.4 Logistic regression model on TFIDF vectorizer

In this step, before I perform text classification using a logistic regression model trained on a term frequency-inverse document frequency (TF-IDF) representation of the text data. This helped me fully consider the frequency of each word in the whole text, improving model accuracy and stability. I initialize a TF-IDF vectorizer object with English stop words, then I got the accuracy and output the top 20 positive words and top 20 negative words by using text_fit function. At this step my accuracy is roundly the same as before, but my coefficients have improved. The accuracy is 93.59%.

- tfidf = TfidfVectorizer(stop_words = 'english')
- text_fit(X, y, tfidf, LogisticRegression())

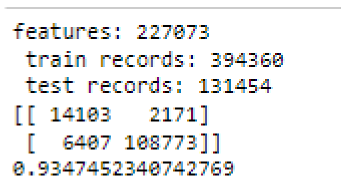


```
features: 227073
 train records: 394360
 test records: 131454
Model Accuracy: 0.9358558887519589
```



```
-Top 20 positive-                          -Top 20 negative-
       Word  Coefficient                              Word  Coefficient
      great    14.289092                              yuck    -5.779824
  delicious    12.475956                            hoping    -5.793518
       best    12.102762                       undrinkable    -5.855910
    perfect    11.258012                             waste    -5.999700
  excellent     9.888044                             worse    -6.263415
      loves     9.663801                             bland    -6.428297
     highly     9.218852                        disgusting    -6.434469
       love     8.550390                             stale    -6.440953
  wonderful     8.512197                              weak    -6.696249
    amazing     8.115916                         tasteless    -6.699791
    awesome     7.928247                            return    -7.045196
       good     7.631838                             threw    -7.210885
   favorite     7.226196                     unfortunately    -7.607589
    pleased     7.156822                          horrible    -8.127590
       nice     7.093360                    disappointment    -8.273381
     hooked     6.974443                      disappointed    -8.683583
      yummy     6.959818                          terrible    -9.024415
 pleasantly     6.704298                             awful    -9.068522
     smooth     6.543144                     disappointing    -9.864559
  fantastic     6.430743                             worst   -12.081919
```

- predict(X, y, tfidf, LogisticRegression())



```
features: 227073
 train records: 394360
 test records: 131454
[[ 14103   2171]
 [  6407 108773]]
0.9347452340742769
```

Then, I got the accuracy and display the Confusion Matrix by prediction function. Accuracy is roughly the same – 93.47%. However, I noticed that the significant words are now more meaningful and have a higher coefficient magnitude.

3.5 Upvote Prediction

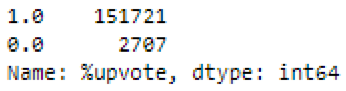
In this step, I am focusing on the score 5 reviews and selecting from its reviews containing 0-20%, 20-40%, 60-80% and 80-100% upvotes and checking the shape of the selected dataset. It contains 154428 data points and 12 features.

- data = df[df['Score'] == 5]
- data2 = data[data['%upvote'].isin(['0-20%', '20-40%', '60-80%', '80-100%'])]

```
data2.shape
```

Then I extract text data from the selected dataset and map upvote to binary output. I'm treating 0-20% and 20-40% as negative category (0), and 60-80% and 80-100% as positive category (1). In the end I came up with a map of upvote tags to a binary output.

```
X = data2['Text']
y_dict = {'0-20%': 0, '20-40%': 0, '60-80%': 1, '80-100%': 1}
y = data2['%upvote'].map(y_dict)
print(y.value_counts())
```



```
1.0    151721
0.0      2707
Name: %upvote, dtype: int64
```

The result of target class 'y' is highly skewed, and I found positive votes are higher than negative votes. This is a case of data imbalance, where using a standard classifier will result in a high error rate,

3.6 Handling imbalanced data using RandomOverSampler

I will re-sample the data to deal with data imbalance and use RandomOverSampler technical. Therefore, I split the data into training and testing sets with 70% of the data used for training and 30% used for testing with the following coding.

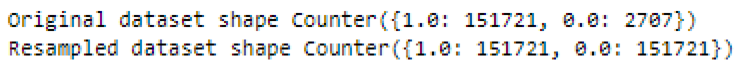
```
tf=TfidfVectorizer()
X_c=tf.fit_transform(X)
X_train,X_test,y_train,y_test=train_test_split(X_c,y,train_size=0.7)
y_test.value_counts()
```

Then, I use RandomOverSampler function from imblearn package to oversample the minority class in the training data to deal with class imbalance. In the following coding, I stored the oversampled data in X_train_res and y_train_res.

- os = RandomOverSampler()
- X_train_res, y_train_res = os.fit_sample(X_c, y)

Here is the result for original and resampled dataset shapes to show the impact of oversampling. It clearly shows the resampled dataset shape counter for negative votes is 151721.

- print('Original dataset shape {}'.format(Counter(y)))
- print('Resampled dataset shape {}'.format(Counter(y_train_res)))



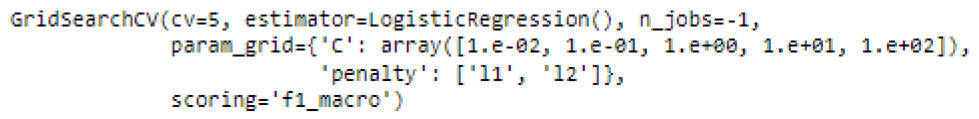
```
Original dataset shape Counter({1.0: 151721, 0.0: 2707})
Resampled dataset shape Counter({1.0: 151721, 0.0: 151721})
```

3.7 Tweaking Parameters

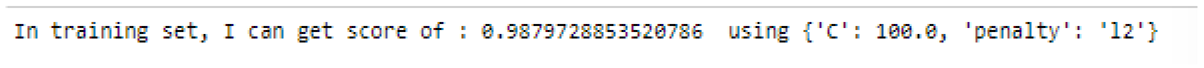
Tweaking parameters in this project is the same as finding the best combination of parameters with previous data mining project. I define a logistic regression model with GridSearchCV() function from sklearn to perform parameters tuning on the model and search the highest parameter combinations with the highest accuracy. In this case, I fit the model on the resampled training data using clf.fit(X_train_res, y_train_res), and then predict the target variable on the test data using y_pred=clf.predict(X_test).

- grid={'C':10.0 **np.arange(-2,3),'penalty':['l1','l2']}
- clf=GridSearchCV(estimator=log_class,param_grid=grid,cv=5,n_jobs=-1,scoring='f1_macro')
- clf.fit(X_train_res,y_train_res)
- y_pred=clf.predict(X_test)
- print(confusion_matrix(y_test,y_pred))
- print(accuracy_score(y_test,y_pred))

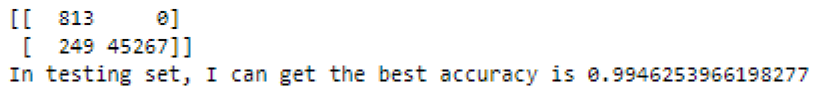
The following result is that I got the best combination of parameters with the highest accuracy; getting a score of 98.80% in the training set, and a score of 99.46% in the testing set. Here is the best combination of parameters, confusion matrix, accuracy score in the training set, and accuracy score of the predictions.



```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': array([1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02]),
                         'penalty': ['l1', 'l2']},
             scoring='f1_macro')
```



```
In training set, I can get score of : 0.9879728853520786  using {'C': 100.0, 'penalty': 'l2'}
```



```
[[  813     0]
 [  249 45267]]
In testing set, I can get the best accuracy is 0.9946253966198277
```

3W

What went well?

Several things went well in this text mining project for sentiment analysis from Amazon reviews.

Firstly, data pre-processing was done efficiently, and the data was cleaned by removing punctuations, URLs, stop words, etc. This helped in reducing the noise in the data and improved the accuracy of the sentiment analysis.

Secondly, the data was explored and visualized using the seaborn library, which provided valuable insights into the data distribution. This helped in identifying patterns in the data and understanding the frequency of words and sentiments in the reviews.

Thirdly, the CountVectorizer function was used to convert the text data into a matrix of token counts, which was then used to train the logistic regression model. The model performed well with an accuracy of 99.41% at the end. The matrix of token counts was fed into the logistic regression model to predict the sentiment of the reviews.

Fourthly, the top positive and negative words were identified using the logistic regression model's coefficients, which helped in understanding the features driving the sentiments.

Finally, the confusion matrix was plotted to evaluate the model's performance. This helped in understanding the model's accuracy and identifying the areas where it was making mistakes.

What did not go well?

One of the major drawbacks of the project was I found the imbalanced data when I predicted upvote. I recognized the target class ‘y’ is higher skewed and positive votes are much higher than negative votes. I initially overlooked the issue. However, after upon reviewing some pieces of knowledge, I found that imbalanced data can have detrimental effects on the performance of a trained model. This is due to the model being biased towards the majority class, leading to poor performance on the minority class and decreased prediction accuracy overall. Moreover, imbalanced data can result in overfitting and bias in the model. It took me some time to learn how to deal with it and found a suitable way to solve this problem.

Also, my computer's CPU and Jupyter memory limitations may be one of the issues running this project. I often get stuck running Jupyter, forcing me to close everything else and reopen it and run it. Especially when it comes to which classifiers to run.

What I would do differently next time?

I learnt how to handle the imbalanced data by using some techniques. In this case, I used RandomOverSampler function to add the number of negative votes. However, I recognized this method might lead to some overfitting issues and I would try other oversampling techniques next time. Also, next time I will deal with imbalanced data before the model, not at the end.

In addition, when I run this type of code later, I will first clean up the computer memory and close unnecessary background software to ensure that the computer will not be stuck because too many things are running at the same time.

I received a fantastic opportunity to learn and advance my abilities through my work using Python-based text mining. I was able to efficiently alter and get important insights from enormous volumes of text data because of Python's flexibility and adaptability. It was thrilling to see how different natural language processing tasks, including sentiment analysis and a bag of words, might be carried out using Python tools like NLTK. It was simpler for me to test out various strategies and algorithms because of the abundance of open-source datasets and resources. My curiosity in text mining has been piqued by this encounter, which has inspired me to look into the field's more cutting-edge uses. Overall, I am happy with the project's results and the insightful information discovered.