DHRUV SRIVASTAVA (USC ID: 8923027371)

CSCI 544 - ASSIGNMENT 1

***Note**: I have explained each step and the function used in the Jupyter notebook and will only talk about the major steps here.*

Step 1: **READING DATA**

1. I am directly reading the data from the given link using “read_csv” function of the pandas library.
2. We read the data directly from the link using the "read_csv" function of pandas. "error_bad_lines=False" is used to Drop any row that contains bad data.
3. We use “dropna()” to drop any row which contains empty/null values.

Step 2: **KEEP REVIEWS & RATINGS**

1. We only need two columns, ('review_body', 'star_rating'), and thus we ignore all the other columns and use only these two.

Python Code: Initial_Dataset = Initial_Dataset[[‘review_body','star_rating']]

Step 3: **LABELLING REVIEWS**

1. Now, we'll add the "Label" column. We'll do labelling in the following way:
    1. if star_rating > 3 = 1
    2. if star_rating < 3 = 0

Python Code: df['label'] = np.where(df[‘star_rating']>3,1,0)

Step 4: **SELECTING 200,000 ENTRIES**

1. Using the "sample" function we'll select 100,000 random values where label = 1 and 100,000 random values where label = 0.

Python Code: positive_ht = df[df.label == 1].sample(100000)
Python Code: negative_ht = df[df.label == 0].sample(100000)

2. After selecting random values, we'll join them together to make a smaller data set which is ready for Data Cleaning and Pre-Processing.

Python Code: data_set = pd.concat([positive_ht,negative_ht])

Step 5: **DATA CLEANING**

1. **Converting all reviews into lower case:**
    - "str.lower( )" is used to convert all characters to lowercase.
    - **Python Code:** data_set['review_body'] = data_set[‘review_body’].str.lower()

2. **Remove HTML from the reviews:**
    - "BeautifulSoup" library is used to extract text from HTML/XML files. We use it here to retrieve just the relevant text and ignore HTML text present in the 'review_body' column.

- **Python Code:** data_set['review_body'] = [BeautifulSoup(X).getText() for X in data_set[‘review_body']]

3. **Remove URL from the reviews:**
   - We used regular expression (" http\S+|www.\S+ ") to filter-out any text which falls under the category of a URL.
   - We "replaced" every URL with a an empty “”
   - **Python Code:** data_set['review_body'] = data_set['review_body'].str.replace('http\S+|www.\S+', '', case=False)

4. **Remove non-alphabetical characters:**
   - The regular expression r'[^a-zA-Z]+' represents all strings that contain non-alphabetical characters and we replaced them with “”.
   - **Python Code:** data_set['review_body'] = data_set.review_body.str.replace(r'[^a-zA-Z]+', '')

5. **Remove Extra space between words:**
   - Used "strip" to remove the extra-spaces at the beginning and the end of a string.
   - **Python Code:** data_set[‘review_body'].str.strip()

   - Used the regular expression ('\s+') to remove the extra space between the words.
   - **Python Code:** data_set['review_body'] = data_set.review_body.str.replace('\s+', ' ', regex=True)

6. **Perform contraction on Reviews:**
   - Used the contractions library to expand all contractions like we'll = we will, you'll = You will.
   - **Python Code:** data_set['review_body'] = data_set['review_body'].apply(lambda x: contractions.fix(x))

Step 6: **PRE-PROCESSING**

1. **Removing Stopwords:**
   - Removed all the stopwords from the 'review_body' column using the "stopwords" list downloaded from NLTK.
   - **Python Code:** nltk.download('stopwords')
   - **Python Code:** stop_words = set(stopwords.words('english'))
   - **Python Code:** data_set['review_body'] = data_set['review_body'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))

2. **Perform Lemmatization**:
   - Lemmatization is the step at which we reduce a word to it's base form. Eg. Studying => Study
   - Wordnet is an large, freely and publicly available lexical database for the English language.
   - We used WhitespaceTokenizer() to extract tokens from string of words without whitespaces etc.
   - We use WordNetLemmatizer() and call the lemmatize() function on each word of the string.
   - The "string_word_lemmetize" function lemmitizes each string (row) passed to it by lemmitizing each word of that string.
   - The "string_word_lemmetize" function returns a list and inorder to convert it into a string, we use the join function.
   - **Python Function:**

```
def string_word_lemmetize(text):
        return [lemmatizer.lemmatize(w) for w in w_tokenizer.tokenize(text)]
```

- **Python Code:** data_set['review_body'] = data_set.review_body.apply(string_word_lemmetize)

Step 6: **SPLITTING DATA INTO TESTING AND TRAINING**

- We split the data into testing and training in the ratio of 20:80.
- **Python Code:** Training_set = data_set.sample(frac = 0.80)
- **Python Code:** Testing_set = data_set.drop(Training_set.index)

Step 7: **TF-IDF FEATURE EXTRACTION**

- TF-IDF stands for Term Frequency – Inverse Document Frequency and we use it for feature extraction.
- The TF-IDF algorithm is implemented using TfidfVectorizer.
- **Python Code:** vectorizer = TfidfVectorizer()

- We usefit_transform( ) method on our training data and transform( ) method on our test data
- We usefit_transform( ) on training data so that we can scale the training data and learn the scaling parameters of that data (Mean, Variance)
- The parameters found using the fit_transform( ) will be used by transform( ) when working on the testing set.
- If we apply fit_transform on testing data as well, then our model would calculate new mean and variance, defeating the purpose of the testing dataset
- **Python Code:** review_body_train_final = vectorizer.fit_transform(review_body_train)
- **Python Code:** review_body_test_final = vectorizer.transform(review_body_test)

Step 8: **RUNNING MODELS & REPORTING Accuracy, Precision, Recall Score & F1 Score**

- Now we have the dataset ready which includes features and labels for the reviews. We have to now just use the data and train different models and test their accuracy. We use 4 different models:
  1. Perceptron (imported from sklearn.linear_model)
  2. SVM (imported from sklearn.svm)
  3. Logistic Regression (imported from sklearn.linear_model)
  4. Multinomial Naive Bayes (imported from sklearn.naive_bayes)
- Used sklearn.metrics for importing:
  - precision_score, recall_score, f1_score and accuracy_score functions.

Example for Perceptron:

```
ppn = Perceptron() #Instantiated the model
ppn.fit(review_body_train_final, label_train) #Trained the model on training set (review_body and label)
pred_ = ppn.predict(review_body_test_final) #Executed the model on the testing set to get it's predicted classification
pred2 = ppn.predict(review_body_train_final) #Executed the model on the training set to get it's predicted classification
```

Printed the Statistcs:

```
print('Testing Data: ',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_test, pred_),precision_score(label_test, pred_), recall_score(label_test, pred_), f1_score(label_test, pred_)))

print('Training Data:',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_train, pred2),precision_score(label_train, pred2), recall_score(label_train, pred2), f1_score(label_train, pred2)))
```

In [1]:

```
!pip install contractions
!pip install sklearn
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import contractions
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from sklearn.model_selection import train_test_split
nltk.download('stopwords')
```

```
Requirement already satisfied: contractions in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (0.0.52)
Requirement already satisfied: textsearch>=0.0.21 in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (fr
om contractions) (0.0.21)
Requirement already satisfied: pyahocorasick in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (from te
xtsearch>=0.0.21->contractions) (1.4.2)
Requirement already satisfied: anyascii in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (from textsea
rch>=0.0.21->contractions) (0.3.0)
Requirement already satisfied: sklearn in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (0.0)
Requirement already satisfied: scikit-learn in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (from skl
earn) (0.24.2)
Requirement already satisfied: scipy>=0.19.1 in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (from sc
ikit-learn->sklearn) (1.6.2)
Requirement already satisfied: joblib>=0.11 in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (from sci
kit-learn->sklearn) (1.0.1)
Requirement already satisfied: numpy>=1.13.3 in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages (from sc
ikit-learn->sklearn) (1.20.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/dhruv/opt/anaconda3/lib/python3.8/site-packages
(from scikit-learn->sklearn) (2.2.0)
[nltk_data] Downloading package wordnet to /Users/dhruv/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/dhruv/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]:

```
True
```

In [2]:

```
#! pip install bs4 # in case you don't have it installed
# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Kitchen_v1_00.tsv.gz
```

1. Read Data

1. We read the data directly from the link using the "read_csv" function of pandas. "error_bad_lines=False" is used to Drop any row that contains bad data.
2. We use "dropna" to drop any row which contains empty/null values.

In [3]:

```
#Initial_Dataset = pd.read_csv("https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Kitchen_v1_0(
Initial_Dataset = pd.read_csv("https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Kitchen_v1_00.
```

```
/Users/dhruv/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3441: FutureWarning: Th
e error_bad_lines argument has been deprecated and will be removed in a future version.


  exec(code_obj, self.user_global_ns, self.user_ns)
b'Skipping line 16148: expected 15 fields, saw 22\nSkipping line 20100: expected 15 fields, saw 22\nSkipping li
ne 45178: expected 15 fields, saw 22\nSkipping line 48700: expected 15 fields, saw 22\nSkipping line 63331: exp
ected 15 fields, saw 22\n'
b'Skipping line 86053: expected 15 fields, saw 22\nSkipping line 88858: expected 15 fields, saw 22\nSkipping li
ne 115017: expected 15 fields, saw 22\n'
b'Skipping line 137366: expected 15 fields, saw 22\nSkipping line 139110: expected 15 fields, saw 22\nSkipping
line 165540: expected 15 fields, saw 22\nSkipping line 171813: expected 15 fields, saw 22\n'
b'Skipping line 203723: expected 15 fields, saw 22\nSkipping line 209366: expected 15 fields, saw 22\nSkipping
line 211310: expected 15 fields, saw 22\nSkipping line 246351: expected 15 fields, saw 22\nSkipping line 25236
4: expected 15 fields, saw 22\n'
b'Skipping line 267003: expected 15 fields, saw 22\nSkipping line 268957: expected 15 fields, saw 22\nSkipping
line 303336: expected 15 fields, saw 22\nSkipping line 306021: expected 15 fields, saw 22\nSkipping line 31156
9: expected 15 fields, saw 22\nSkipping line 316767: expected 15 fields, saw 22\nSkipping line 324009: expected
15 fields, saw 22\n'
b'Skipping line 359107: expected 15 fields, saw 22\nSkipping line 368367: expected 15 fields, saw 22\nSkipping
line 381180: expected 15 fields, saw 22\nSkipping line 390453: expected 15 fields, saw 22\n'
b'Skipping line 412243: expected 15 fields, saw 22\nSkipping line 419342: expected 15 fields, saw 22\nSkipping
line 457388: expected 15 fields, saw 22\n'
b'Skipping line 459935: expected 15 fields, saw 22\nSkipping line 460167: expected 15 fields, saw 22\nSkipping
line 466460: expected 15 fields, saw 22\nSkipping line 500314: expected 15 fields, saw 22\nSkipping line 50033
9: expected 15 fields, saw 22\nSkipping line 505396: expected 15 fields, saw 22\nSkipping line 507760: expected
15 fields, saw 22\nSkipping line 513626: expected 15 fields, saw 22\n'
b'Skipping line 527638: expected 15 fields, saw 22\nSkipping line 534209: expected 15 fields, saw 22\nSkipping
line 535687: expected 15 fields, saw 22\nSkipping line 547671: expected 15 fields, saw 22\nSkipping line 54905
4: expected 15 fields, saw 22\n'
b'Skipping line 599929: expected 15 fields, saw 22\nSkipping line 604776: expected 15 fields, saw 22\nSkipping
line 609937: expected 15 fields, saw 22\nSkipping line 632059: expected 15 fields, saw 22\nSkipping line 63854
6: expected 15 fields, saw 22\n'
b'Skipping line 665017: expected 15 fields, saw 22\nSkipping line 677680: expected 15 fields, saw 22\nSkipping
line 684370: expected 15 fields, saw 22\nSkipping line 720217: expected 15 fields, saw 29\n'
b'Skipping line 723240: expected 15 fields, saw 22\nSkipping line 723433: expected 15 fields, saw 22\nSkipping
line 763891: expected 15 fields, saw 22\n'
b'Skipping line 800288: expected 15 fields, saw 22\nSkipping line 802942: expected 15 fields, saw 22\nSkipping
line 803379: expected 15 fields, saw 22\nSkipping line 805122: expected 15 fields, saw 22\nSkipping line 82189
9: expected 15 fields, saw 22\nSkipping line 831707: expected 15 fields, saw 22\nSkipping line 842829: expected
15 fields, saw 22\nSkipping line 843604: expected 15 fields, saw 22\n'
b'Skipping line 863904: expected 15 fields, saw 22\nSkipping line 875655: expected 15 fields, saw 22\nSkipping
line 886796: expected 15 fields, saw 22\nSkipping line 892299: expected 15 fields, saw 22\nSkipping line 90251
8: expected 15 fields, saw 22\nSkipping line 903079: expected 15 fields, saw 22\nSkipping line 912678: expected
15 fields, saw 22\n'
b'Skipping line 932953: expected 15 fields, saw 22\nSkipping line 936838: expected 15 fields, saw 22\nSkipping
line 937177: expected 15 fields, saw 22\nSkipping line 947695: expected 15 fields, saw 22\nSkipping line 96071
3: expected 15 fields, saw 22\nSkipping line 965225: expected 15 fields, saw 22\nSkipping line 980776: expected
15 fields, saw 22\n'
b'Skipping line 999318: expected 15 fields, saw 22\nSkipping line 1007247: expected 15 fields, saw 22\nSkipping
line 1015987: expected 15 fields, saw 22\nSkipping line 1018984: expected 15 fields, saw 22\nSkipping line 1028
671: expected 15 fields, saw 22\n'
b'Skipping line 1063360: expected 15 fields, saw 22\nSkipping line 1066195: expected 15 fields, saw 22\nSkippin
g line 1066578: expected 15 fields, saw 22\nSkipping line 1066869: expected 15 fields, saw 22\nSkipping line 10
68809: expected 15 fields, saw 22\nSkipping line 1069505: expected 15 fields, saw 22\nSkipping line 1087983: ex
pected 15 fields, saw 22\nSkipping line 1108184: expected 15 fields, saw 22\n'
b'Skipping line 1118137: expected 15 fields, saw 22\nSkipping line 1142723: expected 15 fields, saw 22\nSkippin
g line 1152492: expected 15 fields, saw 22\nSkipping line 1156947: expected 15 fields, saw 22\nSkipping line 11
72563: expected 15 fields, saw 22\n'
b'Skipping line 1209254: expected 15 fields, saw 22\nSkipping line 1212966: expected 15 fields, saw 22\nSkippin
g line 1236533: expected 15 fields, saw 22\nSkipping line 1237598: expected 15 fields, saw 22\n'
b'Skipping line 1273825: expected 15 fields, saw 22\nSkipping line 1277898: expected 15 fields, saw 22\nSkippin
g line 1283654: expected 15 fields, saw 22\nSkipping line 1286023: expected 15 fields, saw 22\nSkipping line 13
02038: expected 15 fields, saw 22\nSkipping line 1305179: expected 15 fields, saw 22\n'
b'Skipping line 1326022: expected 15 fields, saw 22\nSkipping line 1338120: expected 15 fields, saw 22\nSkippin
g line 1338503: expected 15 fields, saw 22\nSkipping line 1338849: expected 15 fields, saw 22\nSkipping line 13
41513: expected 15 fields, saw 22\nSkipping line 1346493: expected 15 fields, saw 22\nSkipping line 1373127: ex
pected 15 fields, saw 22\n'
b'Skipping line 1389508: expected 15 fields, saw 22\nSkipping line 1413951: expected 15 fields, saw 22\nSkippin
g line 1433626: expected 15 fields, saw 22\n'
b'Skipping line 1442698: expected 15 fields, saw 22\nSkipping line 1472982: expected 15 fields, saw 22\nSkippin
g line 1482282: expected 15 fields, saw 22\nSkipping line 1487808: expected 15 fields, saw 22\nSkipping line 15
00636: expected 15 fields, saw 22\n'
b'Skipping line 1511479: expected 15 fields, saw 22\nSkipping line 1532302: expected 15 fields, saw 22\nSkippin
g line 1537952: expected 15 fields, saw 22\nSkipping line 1539951: expected 15 fields, saw 22\nSkipping line 15
41020: expected 15 fields, saw 22\n'
b'Skipping line 1594217: expected 15 fields, saw 22\nSkipping line 1612264: expected 15 fields, saw 22\nSkippin
g line 1615907: expected 15 fields, saw 22\nSkipping line 1621859: expected 15 fields, saw 22\n'
b'Skipping line 1653542: expected 15 fields, saw 22\nSkipping line 1671537: expected 15 fields, saw 22\nSkippin
g line 1672879: expected 15 fields, saw 22\nSkipping line 1674523: expected 15 fields, saw 22\nSkipping line 16
77355: expected 15 fields, saw 22\nSkipping line 1703907: expected 15 fields, saw 22\n'
b'Skipping line 1713046: expected 15 fields, saw 22\nSkipping line 1722982: expected 15 fields, saw 22\nSkippin
g line 1727290: expected 15 fields, saw 22\nSkipping line 1744482: expected 15 fields, saw 22\n'
b'Skipping line 1803858: expected 15 fields, saw 22\nSkipping line 1810069: expected 15 fields, saw 22\nSkippin
g line 1829751: expected 15 fields, saw 22\nSkipping line 1831699: expected 15 fields, saw 22\n'
b'Skipping line 1863131: expected 15 fields, saw 22\nSkipping line 1867917: expected 15 fields, saw 22\nSkippin
g line 1874790: expected 15 fields, saw 22\nSkipping line 1879952: expected 15 fields, saw 22\nSkipping line 18
80501: expected 15 fields, saw 22\nSkipping line 1886655: expected 15 fields, saw 22\nSkipping line 1887888: ex
pected 15 fields, saw 22\nSkipping line 1894286: expected 15 fields, saw 22\nSkipping line 1895400: expected 15
fields, saw 22\n'
b'Skipping line 1904040: expected 15 fields, saw 22\nSkipping line 1907604: expected 15 fields, saw 22\nSkippin
g line 1915739: expected 15 fields, saw 22\nSkipping line 1921514: expected 15 fields, saw 22\nSkipping line 19
39428: expected 15 fields, saw 22\nSkipping line 1944342: expected 15 fields, saw 22\nSkipping line 1949699: ex
pected 15 fields, saw 22\nSkipping line 1961872: expected 15 fields, saw 22\n'
b'Skipping line 1968846: expected 15 fields, saw 22\nSkipping line 1999941: expected 15 fields, saw 22\nSkippin
g line 2001492: expected 15 fields, saw 22\nSkipping line 2011204: expected 15 fields, saw 22\nSkipping line 20
25295: expected 15 fields, saw 22\n'
b'Skipping line 2041266: expected 15 fields, saw 22\nSkipping line 2073314: expected 15 fields, saw 22\nSkippin
g line 2080133: expected 15 fields, saw 22\nSkipping line 2088521: expected 15 fields, saw 22\n'
b'Skipping line 2103490: expected 15 fields, saw 22\nSkipping line 2115278: expected 15 fields, saw 22\nSkippin
g line 2153174: expected 15 fields, saw 22\nSkipping line 2161731: expected 15 fields, saw 22\n'
b'Skipping line 2165250: expected 15 fields, saw 22\nSkipping line 2175132: expected 15 fields, saw 22\nSkippin
g line 2206817: expected 15 fields, saw 22\nSkipping line 2215848: expected 15 fields, saw 22\nSkipping line 22
23811: expected 15 fields, saw 22\n'
b'Skipping line 2257265: expected 15 fields, saw 22\nSkipping line 2259163: expected 15 fields, saw 22\nSkippin
g line 2263291: expected 15 fields, saw 22\n'
b'Skipping line 2301943: expected 15 fields, saw 22\nSkipping line 2304371: expected 15 fields, saw 22\nSkippin
g line 2306015: expected 15 fields, saw 22\nSkipping line 2312186: expected 15 fields, saw 22\nSkipping line 23
14740: expected 15 fields, saw 22\nSkipping line 2317754: expected 15 fields, saw 22\n'
b'Skipping line 2383514: expected 15 fields, saw 22\n'
b'Skipping line 2449763: expected 15 fields, saw 22\n'
b'Skipping line 2589323: expected 15 fields, saw 22\n'
b'Skipping line 2775036: expected 15 fields, saw 22\n'
b'Skipping line 2935174: expected 15 fields, saw 22\n'
b'Skipping line 3078830: expected 15 fields, saw 22\n'
b'Skipping line 3123091: expected 15 fields, saw 22\n'
b'Skipping line 3185533: expected 15 fields, saw 22\n'
b'Skipping line 4150395: expected 15 fields, saw 22\n'
b'Skipping line 4748401: expected 15 fields, saw 22\n'
```

In [4]:

```
Initial_Dataset = Initial_Dataset.dropna()
```

2. Keep Reviews and Ratings

We only need two columns, ('review_body', 'star_rating'), and thus we ignore all the other columns and use only these two.

In [5]:

```
Initial_Dataset = Initial_Dataset[['review_body','star_rating']]
```

Three Sample Reviews and Ratings:

In [6]:

```
print(Initial_Dataset.sample(3))
```

```
                                               review_body  star_rating
2292547  Bought these for my kid's Frozen birthday part...          3.0
3188810  These work great, make the cutest little sandw...          5.0
41726                                Holds bottles perfectly          5.0
```

3. Statistics of Ratings:

In [7]:

```
Initial_Dataset[["star_rating"]].describe()
```

Out[7]:

	star_rating
count	4.874562e+06
mean	4.207322e+00
std	1.286991e+00
min	1.000000e+00
25%	4.000000e+00
50%	5.000000e+00
75%	5.000000e+00
max	5.000000e+00

Reviews grouped by Star Rating received:

In [8]:

```
Agg_Data = Initial_Dataset.groupby(["star_rating"]).count()
Agg_Data = Agg_Data.reset_index()
print(Agg_Data)
# bring count in review_body
```

```
   star_rating  review_body
0          1.0       426852
1          2.0       241931
2          3.0       349533
3          4.0       731693
4          5.0      3124553
```

Count for each class of reviews:

In [9]:

```
Neutral_Reviews  = Initial_Dataset[Initial_Dataset['star_rating']==3]['star_rating'].count()
Positive_Reviews = Initial_Dataset[Initial_Dataset['star_rating']>3]['star_rating'].count()
Negative_Reviews = Initial_Dataset[Initial_Dataset['star_rating']<3]['star_rating'].count()
print('Neutral Reviews: ', Neutral_Reviews,',','Positive Reviews: ',Positive_Reviews,',','Negavtive Reviews: ',
```

```
Neutral Reviews:  349533 , Positive Reviews:  3856246 , Negavtive Reviews:  668783
```

4. Labelling Reviews:

The reviews with rating 4,5 are labelled to be 1 and 1,2 are labelled as 0. Discard the reviews with rating 3'

We don't need to consider "Neutral Reviews" and we'll be ignoring them in the following step.

In [10]:

```
df = Initial_Dataset[Initial_Dataset['star_rating']!=3]
```

Now, we'll add the "Label" column. We'll do labelling in the following way:

1. if star_rating > 3 = 1
2. if star_rating < 3 = 0

In [11]:

```
df['label'] = df['star_rating'].apply(lambda x: 1 if x >3 else 0)
```

```
<ipython-input-11-5e5934107249>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df['label'] = df['star_rating'].apply(lambda x: 1 if x >3 else 0)
```

We select 200000 reviews randomly with 100,000 positive and 100,000 negative reviews.

1. Using the "sample" function we'll select 100,000 random values where label = 1 and 100,000 random values where label = 0.
2. After selecting random values, we'll join them together to make a smaller data set which is ready for Data Cleaning and Pre-Processing.

In [12]:

```
positive_ht = df[df.label == 1].sample(100000)
```

In [13]:

```
negative_ht = df[df.label == 0].sample(100000)
```

In [14]:

```
data_set = pd.concat([positive_ht,negative_ht])
```

Average length of string BEFORE cleaning:

In [15]:

```
temp1 = data_set["review_body"].str.len()
```

In [16]:

```
Avg_len_BeforeCleaning = temp1.mean()
print('Average length of string BEFORE cleaning: ',Avg_len_BeforeCleaning,'\n')
```

```
Average length of string BEFORE cleaning:  322.38637
```

Three sample readings BEFORE data cleaning and pre-processing:

In [17]:

```
print(data_set.sample(3))
```

```
                                               review_body  star_rating  label
2352194                                   Grindsmart coffee          5.0      1
413769   Trim, beautiful look and no plastic!!!  Makes ...          5.0      1
3315622  We visiting some friends and they mentioned th...          5.0      1
```

5. Data Cleaning

5.a) Convert the all reviews into the lower case.

"str.lower( )" is used to convert all characters to lowercase.

In [18]:

```
data_set['review_body'] = data_set['review_body'].str.lower()
```

5.b) remove the HTML and URLs from the reviews

Removing HTML

"BeautifulSoup" library is used to extract text from HTML/XML files. We use it here to retrieve just the relevant text and ignore HTML text present in the 'review_body' column.

In [19]:

```
data_set['review_body']  = [BeautifulSoup(X).getText() for X in data_set['review_body']]
```

```
/Users/dhruv/opt/anaconda3/lib/python3.8/site-packages/bs4/__init__.py:332: MarkupResemblesLocatorWarning: "."
looks like a filename, not markup. You should probably open this file and pass the filehandle into Beautiful So
up.
  warnings.warn(
```

Removing URL

1. We used regular expression (" http\S+|www.\S+ ") to filter-out any text which falls under the category of a URL.
2. We "replaced" every URL with a an empty ""

In [20]:

```
data_set['review_body'] = data_set['review_body'].str.replace('http\S+|www.\S+', '', case=False)
```

```
<ipython-input-20-0b30f0c5afab>:1: FutureWarning: The default value of regex will change from True to False in
a future version.
  data_set['review_body'] = data_set['review_body'].str.replace('http\S+|www.\S+', '', case=False)
```

5.c) remove non-alphabetical characters

The regular expression r'[^a-zA-Z]+' represents all strings that contain non-alphabetical characters and we replaced them with "".

In [21]:

```
data_set['review_body'] = data_set.review_body.str.replace(r'[^a-zA-Z ]+', '')
```

```
<ipython-input-21-55bd09950c89>:1: FutureWarning: The default value of regex will change from True to False in
a future version.
  data_set['review_body'] = data_set.review_body.str.replace(r'[^a-zA-Z ]+', '')
```

5.d) Remove the extra spaces between the words

1. Used "strip" to remove the extra-spaces at the beginning and the end of a string.
2. Used the regular expression ('\s+') to remove the extra space between the words.

In [22]:

```
data_set['review_body'] = data_set['review_body'].str.strip()
data_set['review_body'] = data_set.review_body.str.replace('\s+', ' ', regex=True)
```

5.e) perform contractions on the reviews.

1. Used the contractions library to expand all contrations like we'll = we will, you'll = You will...

In [23]:

```
data_set['review_body'] = data_set['review_body'].apply(lambda x: contractions.fix(x))
```

Average length of string AFTER cleaning OR Average length of string BEFORE Pre-processing :

In [24]:

```
temp2 = data_set["review_body"].str.len()
Average_length_AFTER_cleaning = temp2.mean()
print(Average_length_AFTER_cleaning)
```

```
307.027425
```

6. Pre-processing

6.a) Removing Stopwords

Removed all the stopwords from the 'review_body' column using the "stopwords" list downloaded from NLTK.

In [25]:

```
stop_words = set(stopwords.words('english'))
data_set['review_body'] = data_set['review_body'].apply(lambda x: ' '.join([word for word in x.split() if word
```

6.b) perform lemmatization

1. Lemmatization is the step at which we reduce a word to it's base form. Eg. Studying => Study
2. Wordnet is an large, freely and publicly available lexical database for the English language.
3. We use WordNetLemmatizer() and call the lemmatize() function on each word of the string.
4. The "string_word_lemmetize" function lemmitizes each string (row) passed to it by lemmitizing each word of that string.
5. We used WhitespaceTokenizer() to extract the tokens from string of words without whitespaces etc.

In [26]:

```
import nltk

w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()

def string_word_lemmetize(text):
    return [lemmatizer.lemmatize(w) for w in w_tokenizer.tokenize(text)]
```

In [27]:

```
data_set['review_body'] = data_set.review_body.apply(string_word_lemmetize)
```

The "string_word_lemmetize" function returns a list and inorder to convert it into a string, we use the join function.

In [28]:

```
data_set['review_body'] = data_set['review_body'].str.join(' ')
```

Three sample readings AFTER data cleaning and pre-processing:

In [29]:

```
print(data_set[['review_body','star_rating','label']].sample(3))
```

```
                                               review_body  star_rating  label
4110443  company claim glass break resistant industry w...          2.0      0
292836   bought one marshall dept store exact style mod...          1.0      0
4206954  I kettle year liked size temp gauge looking ke...          2.0      0
```

Average length of string AFTER pre-processing:

In [30]:

```
temp3 = data_set["review_body"].str.len()
Average_length_after_preprocessing = temp3.mean()
print(Average_length_after_preprocessing)
```

```
190.182485
```

DATASET Split 80-20

We split the data into training and testing dataset

In [34]:

```
review_body_train,review_body_test,label_train,label_test = train_test_split(data_set["review_body"], data_set[
```

7. TF-IDF Feature Extraction

1. TF-IDF stands for Term Frequency – Inverse Document Frequency and we use it for feature extraction.
2. The TF-IDF algorithm is implemented using TfidfVectorizer.

In [35]:

```
vectorizer = TfidfVectorizer()
```

1. We usefit_transform( ) method on our training data and transform( ) method on our test data
2. We usefit_transform( ) on training data so that we can scale the training data and learn the scaling parameters of that data (Mean, Variance)
3. The parameters found using the fit_transform( ) will be used by transform( ) when working on the testing set.
4. If we apply fit_transform on testing data as well, then our model would calculate new mean and variance, defeating the purpose of the testing dataset

In [36]:

```
review_body_train_final = vectorizer.fit_transform(review_body_train)
review_body_test_final = vectorizer.transform(review_body_test)
```

Now we have the datset ready which includes features and labels for the reviews. We have to now just use the data and train different models and test their accuracy. We use 4 different models:

1. Perceptron (imported from sklearn.linear_model)
2. SVM (imported from sklearn.svm)
3. Logistic Regression (imported from sklearn.linear_model)
4. Multinomial Naive Bayes (imported from sklearn.naive_bayes)

In [37]:

```
print("\n---------------FINAL OUTPUTS---------------\n")
print('Neutral Reviews: ', Neutral_Reviews,',','Positive Reviews: ',Positive_Reviews,',','Negavtive Reviews: ',
print('Average length of string BEFORE cleaning: ',Avg_len_BeforeCleaning)
print('Average length of string AFTER cleaning: ',Average_length_AFTER_cleaning)
print('Average length of string BEFORE pre-processing: ',Average_length_AFTER_cleaning)
print('Average length of string AFTER cleaning and pre-processing: ',Average_length_after_preprocessing,'\n')
```

```
---------------FINAL OUTPUTS---------------

Neutral Reviews:  349533 , Positive Reviews:  3856246 , Negavtive Reviews:  668783

Average length of string BEFORE cleaning:  322.38637
Average length of string AFTER cleaning:  307.027425
Average length of string BEFORE pre-processing:  307.027425
Average length of string AFTER cleaning and pre-processing:  190.182485
```

7.a) Perceptron

In [38]:

```
ppn = Perceptron()
ppn.fit(review_body_train_final, label_train)
pred_ = ppn.predict(review_body_test_final)
pred2 = ppn.predict(review_body_train_final)
```

In [39]:

```
print('\nPerceptron:')
print('Testing Data: ',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_test, pred_),precision_score(label_test
print('Training Data:',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_train, pred2),precision_score(label_tra
```

```
Perceptron:
Testing Data:  0.85, 0.84, 0.87, 0.86
Training Data: 0.92, 0.91, 0.93, 0.92
```

7.b) SVM

In [40]:

```
lsvc = LinearSVC()
lsvc.fit(review_body_train_final, label_train)
pred_SVM = lsvc.predict(review_body_test_final)
pred2SVM = lsvc.predict(review_body_train_final)
```

In [41]:

```
print('\nSVM:')
print('Testing Data: ',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_test, pred_SVM),precision_score(label_t
print('Training Data:',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_train, pred2SVM),precision_score(label_
```

```
SVM:
Testing Data:  0.89, 0.90, 0.89, 0.89
Training Data: 0.94, 0.94, 0.94, 0.94
```

7.c) Logistic Regression

In [42]:

```
logisticRegr = LogisticRegression(max_iter=1000)
logisticRegr.fit(review_body_train_final, label_train)
pred_LR = logisticRegr.predict(review_body_test_final)
pred2_LR = logisticRegr.predict(review_body_train_final)
```

In [43]:

```
print('\nLogistic Regression:')
print('Testing Data: ',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_test, pred_LR),precision_score(label_te
print('Training Data:',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_train, pred2_LR),precision_score(label_
```

```
Logistic Regression:
Testing Data:  0.90, 0.90, 0.89, 0.90
Training Data: 0.91, 0.92, 0.91, 0.91
```

7.d) Multinomial Naive Bayes

In [44]:

```
nb = MultinomialNB()
nb.fit(review_body_train_final, label_train)
pred_NB = nb.predict(review_body_test_final)
pred2_NB = nb.predict(review_body_train_final)
```

In [45]:

```
print('\nNaive Bayes:')
print('Testing Data: ',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_test, pred_NB),precision_score(label_te
print('Training Data:',"%.2f, %.2f, %.2f, %.2f" % (accuracy_score(label_train, pred2_NB),precision_score(label_
```

```
Naive Bayes:
Testing Data:  0.87, 0.89, 0.85, 0.87
Training Data: 0.89, 0.90, 0.88, 0.89
```