Libraries

The following libraries might be required for the .py file to function properly

- scikit-learn
- nltk
- contractions
- pandas
- numpy
- bs4

In [2]:
```
! pip install bs4
! pip install contractions
# Dataset: https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/amazon
```

```
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: bs4 in c:\users\bhavi\anaconda3\lib\site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in c:\users\bhavi\anaconda3\lib\site-packa
ges (from bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\bhavi\anaconda3\lib\site-packag
es (from beautifulsoup4->bs4) (2.3.1)
[notice] A new release of pip available: 22.3.1 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: contractions in c:\users\bhavi\anaconda3\lib\site-package
s (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in c:\users\bhavi\anaconda3\lib\site-p
ackages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in c:\users\bhavi\anaconda3\lib\site-packages (f
rom textsearch>=0.0.21->contractions) (0.3.2)
Requirement already satisfied: pyahocorasick in c:\users\bhavi\anaconda3\lib\site-packag
es (from textsearch>=0.0.21->contractions) (2.0.0)
[notice] A new release of pip available: 22.3.1 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [ ]:
```
import os # for checking if file is present
from urllib import request # for downloading the dataset
import gzip # for extracting the dataset

import pandas as pd
import numpy as np
import nltk # for pre-processing tasks like tokenization, stop words removal, and lemmat
import re # for removing urls, extra spaces etc.
from bs4 import BeautifulSoup # for removal of html
import contractions # for expanding contractions

from nltk.corpus import wordnet, stopwords
from nltk import pos_tag # pos_tagging to be used in conjunction with lemmatizer
from nltk.stem import WordNetLemmatizer # lemmatizer

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer # for creat
from sklearn.model_selection import train_test_split # for splitting into training and t
from sklearn.metrics import precision_score, recall_score, f1_score  # for calculating m

# models to be used for training
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
```

```
# downloading the different requirements for using nltk pos_tag, stop words and wordnet
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

Read Data

- First, we read the data. To do this, we make use of urllib.request library. We retrieve the file from the dataset url provided and then store it locally.
- Once the data is downloaded, we extract it from the gzipped file and save a .tsv version.
- This data can be then read using pd.read_csv or pd.read_table.
- We use '\t' as the separator as it is a .tsv file.
- While trying to create the data frame, there were errors where we had 21 columns instead of 15, so on_bad_lines was set to 'skip'.

In [3]:
```
url = 'https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/amazon-rev

extracted_file = 'amazon_reviews_us_Office_Products.tsv'
compressed_file = extracted_file + '.gz'


# Retrieve the dataset from given url and store it in location specified by compressed_f
if not os.path.exists(extracted_file):
    request.urlretrieve(url, compressed_file)

    # extract the dataset from the gzipped file
    with gzip.open(compressed_file, 'rb') as f_in, open(extracted_file, 'wb') as f_out:
        for line in f_in:
            f_out.write(line)

    os.remove(compressed_file)

# read the extracted data into pandas dataframe
original_df = pd.read_csv(extracted_file, sep='\t', on_bad_lines='skip', low_memory=Fals
print(original_df.head())
```

Keep Reviews and Ratings

- Now, we try to save only two columns: review_body and star_rating.
- Here, I noticed that some of the values in star_rating included dates, which was unexpected.
- Since these were erroneous, I decided to drop them by converting the column to numeric and coercing any errors, which will turn them to NaN values.

In [4]:
```
# creating the dataframe by taking only review_body and star_rating columns
df = pd.DataFrame(original_df[['review_body', 'star_rating']])
print(df.head())

# we notice there are some erroneous values for the star_rating column
print(df['star_rating'].unique())

# converting the star_rating to numeric values and dropping erroneous columns
df['star_rating'] = pd.to_numeric(df['star_rating'], errors='coerce')
df.dropna(inplace=True)

print(df['star_rating'].unique())
```

```
                                         review_body star_rating
```

```
0                                       Great product.         5
1  What's to say about this commodity item except...           5
2    Haven't used yet, but I am sure I will like it.           5
3  Although this was labeled as &#34;new&#34; the...           1
4                      Gorgeous colors and easy to use         4
['5' '1' '4' '2' '3' '2015-06-05' '2015-02-11' nan '2014-02-14']
[5. 1. 4. 2. 3.]
```

We form two classes and select 50000 reviews randomly from each class.

- Now, a new column called target is created, where there are only two values: 1 and 2. 1 is given to star_rating rows with values 1, 2 or 3, and 2 is given to star_rating rows with values 4 or 5.
- Afterwards, 50000 rows of each target class 1 or 2 are sampled into two different intermediate variables: class_1 and class_2.
- Finally, a new dataframe is created concatenating these two intermediate variables.

In [5]:

```python
# creating the target column: target = 1 if star_rating is 1, 2 or 3. target = 2 if star_
df['star_rating'] = df['star_rating'].astype(int)
df['target'] = df['star_rating'].apply(lambda x: 1 if x <= 3 else 2)

sample_size = 50000

# creating a sample dataframe where target = 1 of size 50000 rows
class_1 = df.loc[df['target'] == 1].sample(n=sample_size, random_state=42)

# creating a sample dataframe where target = 2 of size 50000 rows
class_2 = df.loc[df['target'] == 2].sample(n=sample_size, random_state=42)

# merging the two sample dataframes
df_new = pd.concat([class_1, class_2], ignore_index=True)
```

```
[2 1]
```

Data Cleaning

Pre-processing

- In cleaning of the data, we perform the following steps inside the clean() function:
  - Converting to lower case: we use the string’s lower() method
  - Removing html: Beautiful soup is used to perform this task. We use the decompose() method to remove any anchor tags which will contain html
  - Removing urls: Urls are removed using regular expressions. This works for both http and https urls.
  - Removing non-alphabetical characters: Non-alphabetical characters are removed by using regular expressions as well.
  - Removal of extra spaces: Extra spaces can be removed by substituting any multiple spaces denoted by ‘\s+’ with a single space.
  - Expanding contractions: The contractions library is used to expand any contractions found within a review. We can use the fix() method to perform the expansion.
- Before performing this data cleaning, the average character length of each review is 314.24925 and after data cleaning, it decreases to 298.3743.

In [6]:
```
import re
import contractions

# convert to lower-case
# remove html and urls
# remove non-alphabetical character
# remove extra spaces
# perform contractions

def clean(review):
    # converting to lowercase
    review = review.lower()

    # removing htmls
    soup = BeautifulSoup(review, "html.parser")

    for a_tag in soup.find_all("a"):
        a_tag.decompose()

    review = soup.get_text()

    # removing urls
    review = re.sub(r'^https?:\/\/.*[\r\n]*', '', review)

    # removing non-alphabetical characters
    review = re.sub(r'[^a-zA-Z\s]', '', review)

    # removing extra spaces
    review = re.sub(r'\s+', ' ', review).strip()

    # expanding contractions
    review = contractions.fix(review)

    return review

# calculating average character length of each review before cleaning
before_cleaning = df_new['review_body'].apply(len).mean()

df_new['review_body'] = df_new['review_body'].apply(clean)

# calculating average character length of each review after cleaning
after_cleaning = df_new['review_body'].apply(len).mean()

print('Average length of reviews before Cleaning: ', before_cleaning, ', Average length
```

```
C:\Users\bhavi\AppData\Local\Temp\ipykernel_27320\2550330846.py:15: MarkupResemblesLocat
orWarning: The input looks more like a filename than markup. You may want to open this f
ile and pass the filehandle into Beautiful Soup.
  soup = BeautifulSoup(review, "html.parser")
C:\Users\bhavi\AppData\Local\Temp\ipykernel_27320\2550330846.py:15: MarkupResemblesLocat
orWarning: The input looks more like a URL than markup. You may want to use an HTTP clie
nt like requests to get the document behind the URL, and feed that document to Beautiful
Soup.
  soup = BeautifulSoup(review, "html.parser")
Average length of reviews before Cleaning: 314.24925, Average length of reviews after cl
eaning: 298.3743
```

remove the stop words

- We use nltk to remove stop words
- We can obtain the set of stop words in english language from nltk.corpus.
- First, we tokenize the words in the review and the token is only included in the output to be returned if it is not present in the set of stopwords.

- In this way, we obtain all the words which are not in stop words
- Before stop words removal we have approximately 298 characters per review which decreases to 188.39753 characters per review.

In [7]:
```
from nltk.corpus import stopwords

def remove_stopwords(review):
    # tokenizing words from the review
    words = nltk.word_tokenize(review)

    # obtaining the set of stop words
    stop = set(stopwords.words('english'))

    # not picking the word if it is present in the set of stop words
    words = [word for word in words if word not in stop]
    review = ' '.join(words)

    return review


# average character length of each review before removing stop words
before_stop_words = df_new['review_body'].apply(len).mean()

df_new['review_body'] = df_new['review_body'].apply(remove_stopwords)

after_stop_words = df_new['review_body'].apply(len).mean()

print('Average length of reviews before removing stop words: ', before_stop_words, ' Ave
```

```
Average length of reviews before removing stop words: 298.3743 Average length of reviews
after removing stop words: 188.39753
```

perform lemmatization

- To perform lemmatization, we can use WordNetLemmatizer from nltk.stem.
- However, it lemmatizes a word based on its part-of-speech which by default is considered as Noun.
- In order to make the lemmatization more accurate, we have to provide its pos tag. We can do this by using the pos_tag() method from nltk.
- However, this provides treebank tags, which need to be converted to Word Net compatible tags.
- This conversion is done by first getting the treebank tags inside the lemmatize function. Then, we call the get_tag() function which converts a treebank tag to wordnet tag. The tag conversion is as follows:
  - A treebank tag beginning with 'J' is an adjective
  - A treebank tag beginning with 'V' is a verb
  - A treebank tag beginning with 'N' is a noun
  - A treebank tag beginning with 'R' is a adverb
- After lemmatization, we notice the average character length drop further to 185.27033 characters per review.

In [8]:
```
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
from nltk.corpus import wordnet

def get_tag(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
```

```
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return ''

def lemmatize(review):
    # tokenizing a review
    words = nltk.word_tokenize(review)

    # creating tags for the review by obtaining the treebank tags and then converting to
    treebank_tags = pos_tag(words)
    tags = [get_tag(word) for word in words]

    lemmatizer = WordNetLemmatizer()

    # We lemmatize the words along with the tag if it is available, else use the default
    lemmatized_words = [lemmatizer.lemmatize(word, tag) if tag != '' else lemmatizer.lem
    review = ' '.join(lemmatized_words)

    return review


# average length of characters per review before lemmatization
before_lemm = df_new['review_body'].apply(len).mean()

df_new['review_body'] = df_new['review_body'].apply(lemmatize)

# average length of characters per review after lemmatization
after_lemm = df_new['review_body'].apply(len).mean()

print('Average length of characters before lemmatization: ', before_lemm, 'Average lengt
```

```
Average length of characters before lemmatization:      188.39753       Average length o
f characters after lemmatization:       185.27033
```

In [9]:
```
print('Average length of reviews before pre-processing: ', before_stop_words, ' ,Average
```

```
Average length of reviews before pre-processing:        298.3743         ,Average length
of reviews after pre-processing:        185.27033
```

TF-IDF and BoW Feature Extraction

- The next task is to extract TF-IDF and Bag-of-Words features.
- We can use sklearn’s CountVectorizer (for bow) and TfidfVectorizer (for tf-idf) classes present in feature_extraction.text library in sklearn.
- We use the fit_transform methods of both classes to obtain the numerical features
- We also split the tf-idf matrix, bow matrix and the target column in df_new in a single step using train_test_split function from sklearn.model_selection. This will help maintain correspondence between not only tf-idf matrix and target, and bow and target, but also tf-idf and bow.

In [10]:
```
# Creating the Bag-of-Words dataset
bow_extractor = CountVectorizer()
bow_matrix = bow_extractor.fit_transform(df_new['review_body'])

# Creating the TF-IDF Dataset
tf_idf_extractor = TfidfVectorizer()
tf_idf_matrix = tf_idf_extractor.fit_transform(df_new['review_body'])

# creating the train and test sets for Bag-of-Words, TF-IDF and targets column
bow_X_train, bow_X_test, tf_idf_X_train, tf_idf_X_test, Y_train, Y_test = train_test_spl
```

Perceptron Using Both Features

Performance for perceptron for bag of words:

- Precision: 0.8156452416542103
- Recall: 0.7908212560386474
- F1-Score: 0.8030414520480745

Performance for perceptron for tf-idf:

- Precision: 0.8338814150473344
- Recall: 0.7725258493353028
- F1-Score: 0.8020319164230604

In [72]:

```
# Creating and training the perceptron for bag-of-words
bow_clf = Perceptron(penalty='elasticnet', l1_ratio=0.1, eta0=1e-3, alpha=1e-6, tol=1e-4
bow_clf.fit(bow_X_train, Y_train)

# making predictions on the bag-of-words test set
bow_Y_pred = bow_clf.predict(bow_X_test)

# calculating and printing the precision, recall and f1 scores for perceptron on the bag
bow_precision = precision_score(bow_Y_pred, Y_test)
bow_recall = recall_score(bow_Y_pred, Y_test)
bow_f1 = f1_score(bow_Y_pred, Y_test)
print('BOW: ', bow_precision, bow_recall, bow_f1)

# Creating and training the perceptron for TF-IDF
tf_idf_clf = Perceptron(penalty='elasticnet', l1_ratio=0.3, eta0=1e-5, max_iter=1000, al
tf_idf_clf.fit(tf_idf_X_train, Y_train)

# making predictions on the TD-IDF test set
tf_idf_Y_pred = tf_idf_clf.predict(tf_idf_X_test)

# calculating and printing the Precision, Recall and F1 Scores for perceptron on the TF-
tf_idf_precision = precision_score(tf_idf_Y_pred, Y_test)
tf_idf_recall = recall_score(tf_idf_Y_pred, Y_test)
tf_idf_f1 = f1_score(tf_idf_Y_pred, Y_test)
print('TF-IDF: ', tf_idf_precision, tf_idf_recall, tf_idf_f1)
```

```
BOW:  0.8156452416542103 0.7908212560386474 0.8030414520480745
TF-IDF:  0.8338814150473344 0.7725258493353028 0.8020319164230604
```

SVM Using Both Features

Performance for SVM for bag of words:

- Precision: 0.820627802690583
- Recall: 0.8598726114649682
- F1-Score: 0.8397919641036101

Performance for SVM for tf-idf:

- Precision: 0.8589935226706528
- Recall: 0.8424550430023455

- F1-Score: 0.8506439038831598

In [173…

```
# Creating and training SVM model on Bag-of-words training set
bow_svm = LinearSVC(max_iter=1000, penalty='l1', dual=False, C=0.1, random_state=42)
bow_svm.fit(bow_X_train, Y_train)

# making predictions on bag-of-words test set
bow_Y_pred = bow_svm.predict(bow_X_test)

# calculating and printing the precision, recall and f1-scores for svm on bag-of-words t
bow_precision = precision_score(bow_Y_pred, Y_test)
bow_recall = recall_score(bow_Y_pred, Y_test)
bow_f1 = f1_score(bow_Y_pred, Y_test)
print('BOW: ', bow_precision, bow_recall, bow_f1)


# Creating and training SVM model on TFIDF training set
tf_idf_svm = LinearSVC(max_iter=10, dual=False, C=0.1, random_state=42)
tf_idf_svm.fit(tf_idf_X_train, Y_train)

# making predictions on TF-IDF test set
tf_idf_Y_pred = tf_idf_svm.predict(tf_idf_X_test)

# calculating and printing the precision, recall and f1-scores for svm on tf-idf test se
tf_idf_precision = precision_score(tf_idf_Y_pred, Y_test)
tf_idf_recall = recall_score(tf_idf_Y_pred, Y_test)
tf_idf_f1 = f1_score(tf_idf_Y_pred, Y_test)
print('TF-IDF: ', tf_idf_precision, tf_idf_recall, tf_idf_f1)
```

```
BOW:  0.820627802690583 0.8598726114649682 0.8397919641036101
TF-IDF:  0.8589935226706528 0.8424550430023455 0.8506439038831598
```

Logistic Regression Using Both Features

Performance for logistic regression for bag of words:

- Precision: 0.8260089686098655
- Recall: 0.8555062441944473
- F1-Score: 0.8404988846075846

Performance for logistic regression for tf-idf:

- Precision: 0.8579970104633782
- Recall: 0.8422185268512179
- F1-Score: 0.8500345542501728

In [176…

```
# Creating and training a Logistic Regression model on Bag-of-words training set
bow_log_reg = LogisticRegression(max_iter=1000, C=0.3, random_state=42)
bow_log_reg.fit(bow_X_train, Y_train)

# making predictions on bag-of-words test set
bow_Y_pred = bow_log_reg.predict(bow_X_test)

# calculating and printing precision, recall and f1-scores for logistic regression on ba
bow_precision = precision_score(bow_Y_pred, Y_test)
bow_recall = recall_score(bow_Y_pred, Y_test)
bow_f1 = f1_score(bow_Y_pred, Y_test)
print('BOW: ', bow_precision, bow_recall, bow_f1)
```

```python
# Creating and training a Logistic Regression model on TF-IDF training set
tf_idf_log_reg = LogisticRegression(max_iter=200, random_state=42)
tf_idf_log_reg.fit(tf_idf_X_train, Y_train)

# making predictions on TF-IDF test set
tf_idf_Y_pred = tf_idf_log_reg.predict(tf_idf_X_test)


# calculating and printing precision, recall and f1-scores for logistic regression on tf
tf_idf_precision = precision_score(tf_idf_Y_pred, Y_test)
tf_idf_recall = recall_score(tf_idf_Y_pred, Y_test)
tf_idf_f1 = f1_score(tf_idf_Y_pred, Y_test)
print('TF-IDF: ', tf_idf_precision, tf_idf_recall, tf_idf_f1)
```

```
BOW:  0.8260089686098655 0.8555062441944473 0.8404988846075846
TF-IDF:  0.8579970104633782 0.8422185268512179 0.8500345542501728
```

Naive Bayes Using Both Features

Performance for naive bayes for bag of words:

- Precision: 0.837767812655705
- Recall: 0.788575180564675
- F1-Score: 0.8124275222265173

Performance for naive bayes for tf-idf:

- Precision: 0.8497259591429995
- Recall: 0.800732463142079
- F1-Score: 0.8245020305550185

In [175…

```python
# Creating and training a Naive-bayes model on Bag-of-words training set
bow_nb = MultinomialNB(alpha=5, force_alpha=True)
bow_nb.fit(bow_X_train, Y_train)

# making predictions on bag-of-words test set
bow_Y_pred = bow_nb.predict(bow_X_test)

# calculating and printing precision, recall and f1-scores for naive bayes on bag-of-wor
bow_precision = precision_score(bow_Y_pred, Y_test)
bow_recall = recall_score(bow_Y_pred, Y_test)
bow_f1 = f1_score(bow_Y_pred, Y_test)
print('BOW: ', bow_precision, bow_recall, bow_f1)


# creating and training a Naive-bayes model on TF-IDF training set
tf_idf_nb = MultinomialNB(alpha=1)
tf_idf_nb.fit(tf_idf_X_train, Y_train)

# making predictions on tf-idf test set
tf_idf_Y_pred = tf_idf_nb.predict(tf_idf_X_test)

# calculating and printing precision, recall and f1-scores for naive bayes on tf-idf tes
tf_idf_precision = precision_score(tf_idf_Y_pred, Y_test)
tf_idf_recall = recall_score(tf_idf_Y_pred, Y_test)
tf_idf_f1 = f1_score(tf_idf_Y_pred, Y_test)
print('TF-IDF: ', tf_idf_precision, tf_idf_recall, tf_idf_f1)
```

```
BOW:  0.837767812655705 0.788575180564675 0.8124275222265173
TF-IDF:  0.8497259591429995 0.800732463142079 0.8245020305550185
```