```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import subprocess
subprocess.run(["pip", "install", "requests"])
subprocess.run(["pip", "install", "nltk"])
subprocess.run(["pip", "install", "scikit-learn"])


# In[2]:


import pandas as pd
import numpy as np
import sklearn
import requests
import io
import nltk
nltk.download('wordnet')


# ## Dataset Preparation

# In[3]:


url = 'https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Office_Products_v1_00.
tsv.gz'
content = requests.get(url).content
df = pd.read_csv(io.BytesIO(content), compression='gzip', sep='\t', on_bad_lines='skip', usecols=['review_body', 'review_headline',
'star_rating'])

print("Three sample review rows:")
print(df.sample(3))


# In[67]:


#convert star_rating to numeric data
df = df[pd.to_numeric(df['star_rating'], errors='coerce').notnull()]
df['star_rating'] = df['star_rating'].astype(float)

print("")
print("Star_rating:")
print("1.0: " + str(df[df['star_rating']==1].shape[0]))
print("2.0: " + str(df[df['star_rating']==2].shape[0]))
print("3.0: " + str(df[df['star_rating']==3].shape[0]))
print("4.0: " + str(df[df['star_rating']==4].shape[0]))
print("5.0: " + str(df[df['star_rating']==5].shape[0]))

#positive reviews
positive_reviews = df[df['star_rating']>3].shape[0]

#neutral reviews
neutral_reviews = df[df['star_rating']==3].shape[0]

#negative reviews
negative_reviews = df[df['star_rating']<3].shape[0]

print("")
print("Positive(label 1), Neutral(label 0) and Negative reviews count:")
print("1 " + str(positive_reviews))
print("0 " + str(neutral_reviews))
print("Neutral " + str(negative_reviews))


# In[5]:


# Adding labels
labeled_data = df
conditions = [(labeled_data['star_rating'] >3), (labeled_data['star_rating'] < 3)]
values = [1, 0]
labeled_data['labels'] = np.select(conditions, values)
#print(labeled_data[['review_headline', 'review_body', 'star_rating', 'labels']].head())


# In[6]:


# Remove duplicate rows and drop null values for review_body or review_headline features
num_of_rows_before = labeled_data.shape[0]
```

```
labeled_data = labeled_data.drop_duplicates(subset=['review_body', 'review_headline'], keep='first').dropna(subset=['review_body']).dropna
(subset=['review_headline'])
num_of_rows_after = labeled_data.shape[0]

#print("Number of rows before and after removing duplicates and null values: ")
#print(str(num_of_rows_before) + "," + str(num_of_rows_after))


# In[7]:


# sample 100000 data
num_of_samples = 100000

positive_reviews = labeled_data[labeled_data.star_rating>3].sample(num_of_samples)
negative_reviews = labeled_data[labeled_data.star_rating<3].sample(num_of_samples)

reduced_labeled_data = pd.concat([positive_reviews, negative_reviews]).sample(frac=1)
#print(reduced_labeled_data.shape)


# ## Dataset Cleaning

# In[8]:


contractions = {
    "'s": "is",
    "'S": "Is",
    "aren't": "are not",
    "arent": "are not",
    "can't": "can not",
    "cant": "can not",
    "can't've": "can not have",
    "'cause": "because",
    "cannot": "can not",
    "could've": "could have",
    "couldve": "could have",
    "couldn't": "could not",
    "couldnt": "could not",
    "couldn't've": "could not have",
    "couldntve": "could not have",
    "didn't": "did not",
    "didnt": "did not",
    "doesn't": "does not",
    "doesnt": "does not",
    "don't": "do not",
    "dont": "do not",
    "hadn't": "had not",
    "hadnt": "had not",
    "hadn't've": "had not have",
    "hasn't": "has not",
    "hasnt": "has not",
    "haven't": "have not",
    "havent": "have not",
    "he'd": "he would",
    "hed": "he would",
    "he'd've": "he would have",
    "hedve": "he would have",
    "he'll": "he will",
    "he'll've": "he will have",
    "he's": "he is",
    "hes": "he is",
    "how'd": "how did",
    "howd": "how did",
    "how'd'y": "how did you",
    "how'll": "how will",
    "howll": "how will",
    "how's": "how is",
    "hows": "how is",
    "i'd": "i would",
    "i'd've": "i would have",
    "i'll": "i will",
    "i'll've": "i will have",
    "i'm": "i am",
    "im": "i am",
    "i'ma": "i am going to",
    "i've": "i have",
    "isn't": "is not",
    "isnt": "is not",
    "it'd": "it would",
    "it'd've": "it would have",
    "it'll've": "it will have",
    "it'll": "it will",
    "itll": "it will",
    "it's": "it is",
    "let's": "let us",
```

```
"lets": "let us",
"ma'am": "madam",
"mayn't": "may not",
"mightn't": "it might not",
"mightn't've": "might not have",
"might've": "might have",
"mustn't": "must not",
"mustn't've": "must not have",
"must've": "must have",
"needn't": "need not",
"needn't've": "need not have",
"not've": "not have",
"oughtn't": "ought not",
"oughtn't've": "ought not to have",
"so've": "so have",
"so's": "so is",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she would",
"she'd've": "she would have",
"she'll": "she will",
"she'll've": "she will have",
"she's": "she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"that'd": "that would",
"that'd've": "that would have",
"that's": "that is",
"thats": "that is",
"there'd": "there would",
"there'd've": "there would have",
"there's": "there is",
"they'd": "they would",
"they'd've": "they would have",
"they'll": "they will",
"they'll've": "they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what will",
"what'll've": "what will have",
"what're": "what are",
"what's": "what has/is",
"what've": "what have",
"when's": "when is",
"when've": "when have",
"where'd": "where would",
"where's": "where is",
"where've": "where have",
"who'd": "who would",
"who'll": "who will",
"who'll've": "who will have",
"who're": "who are",
"who's": "who is",
"who've": "who have",
"why've": "why have",
"why'll": "why will",
"why're": "why are",
"why's": "why is",
"will've": "will have",
"won't": "will not",
"wont": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you would",
"you'd've": "you would have",
"you'll": "you will",
"you'll've": "you will have",
"you're": "you are",
"you've": "you have"
```

```
}
```

```
# In[9]:
```

```
reduced_labeled_data['review'] = reduced_labeled_data['review_headline'] + ' ' + reduced_labeled_data['review_body']

review_mean_before = reduced_labeled_data['review'].str.len().mean()

# to lower case
reduced_labeled_data['review'] = reduced_labeled_data['review'].astype(str).map(str.lower)

# remove extra spaces
reduced_labeled_data['review'] = reduced_labeled_data['review'].str.replace(r' +|\t+', ' ', regex=True)

# remove html characters
reduced_labeled_data['review'] = reduced_labeled_data['review'].str.replace('<[^<>]*>', '', regex=True)

#remove urls, https
reduced_labeled_data['review'] = reduced_labeled_data['review'].str.replace(r'http\S+|www.\S+', '', case=False, regex=True)

#expanding contractions
reduced_labeled_data['review'] = reduced_labeled_data['review'].map(lambda review: ' '.join(contractions.get(word, word) for word in review.
split(' ')))

#remove non-alphabetical characters
reduced_labeled_data['review'] = reduced_labeled_data['review'].str.replace(r'[^a-zA-Z\s]', '', regex=True).str.replace(r'[^\w\s]', '', regex=
True).str.replace(r'\d', '', regex=True)

review_mean_after = reduced_labeled_data['review'].str.len().mean()

print("")
print("Review mean length before and after data processing:" + str(review_mean_before) + ", " + str(review_mean_after))
#print("Review data after cleaning based on regex:")
#print(reduced_labeled_data[['review', 'labels']].head())
```

```
# ## Preprocessing
```

```
# In[10]:
```

```
from nltk.corpus import stopwords

nltk.download('stopwords')
stop_words = set(stopwords.words('English'))

review_mean_before = reduced_labeled_data['review'].str.len().mean()
reduced_labeled_data['review']=reduced_labeled_data['review'].map(lambda review: ' '.join(word for word in review.split(" ") if not word in
stop_words))
review_mean_after = reduced_labeled_data['review'].str.len().mean()

print("")
print("Review mean length before and after removing stop words:" + str(review_mean_before) + "," + str(review_mean_after))
#print("Review data after removing stop words:")
#print(reduced_labeled_data[['review', 'labels']].head())
```

```
# In[11]:
```

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
review_mean_before = reduced_labeled_data['review'].str.len().mean()
reduced_labeled_data['review'] = reduced_labeled_data['review'].map(lambda review: ' '.join(lemmatizer.lemmatize(word) for word in review.split
(' ')))
review_mean_after = reduced_labeled_data['review'].str.len().mean()

print("Review mean length before and after lemmatization:" + str(review_mean_before) + "," + str(review_mean_after))
#print("Review data after lemmatization:")
#print(reduced_labeled_data[['review', 'labels']].head())
```

```
# ## Feature Extraction
```

```
# In[12]:
```

```
from sklearn.model_selection import train_test_split

datasets = train_test_split(reduced_labeled_data['review'], reduced_labeled_data['labels'], test_size = 0.2)
train_data, test_data, train_labels, test_labels = datasets

#print("Number of train and test data sets:")
#print(str(len(train_data)) + ", " + str(len(test_data)))
```

```
# In[51]:


from sklearn.feature_extraction.text import TfidfVectorizer

train_vectorizer = TfidfVectorizer(max_features = 25000)#, sublinear_tf=True, max_df=0.3, min_df=10, ngram_range = (1,4), stop_words="
english", strip_accents='ascii')
train_features = train_vectorizer.fit_transform(train_data)
test_features = train_vectorizer.transform(test_data)

#print("Train features and test features printed below:")
#print(train_features)
#print("")
#print(test_features)


# In[61]:


def printMatrix(matrix):
    print("Accuracy: " + str(matrix['accuracy']))
    print("Precision: " + str(matrix['macro avg']['precision']))
    print("Recall: " + str(matrix['macro avg']['recall']))
    print("F1-score: " + str(matrix['macro avg']['f1-score']))
    print("")


# ## Perceptron

# In[53]:


#perceptron training

from sklearn.linear_model import Perceptron

p = Perceptron(random_state = 42)
p.fit(train_features, train_labels)


# In[62]:


# Metrics
from sklearn.metrics import accuracy_score, classification_report

train_predictions = p.predict(train_features)
test_predictions = p.predict(test_features)

print("Perceptron Training Metrics:")
printMatrix(classification_report(train_predictions, train_labels, output_dict=True))
print("")
print("Perceptron Testing Metrics:")
printMatrix(classification_report(test_predictions, test_labels, output_dict=True))


# ## SVM

# In[55]:


# SVM classifier

from sklearn.svm import LinearSVC

svm_classifier = LinearSVC()
svm_classifier.fit(train_features, train_labels)


# In[63]:


#SVM metrics
train_predictions = svm_classifier.predict(train_features)
test_predictions = svm_classifier.predict(test_features)

print("SVM Training Metrics:")
printMatrix(classification_report(train_predictions, train_labels, output_dict=True))
print("")
print("SVM Testing Metrics:")
printMatrix(classification_report(test_predictions, test_labels, output_dict=True))


# ## Logistic Regression
```

```
# In[57]:


# Logistic Regression
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(random_state = 41)
LR.fit(train_features, train_labels)


# In[64]:


#LR metrics

train_predictions = LR.predict(train_features)
test_predictions = LR.predict(test_features)

print("Logistic Regression Training Metrics:")
printMatrix(classification_report(train_predictions, train_labels, output_dict=True))
print("")
print("Logistic Regression Testing Metrics:")
printMatrix(classification_report(test_predictions, test_labels, output_dict=True))


# ## Multinomial Naive Bayes

# In[59]:


# Naive Bayes
from sklearn.naive_bayes import MultinomialNB

NB = MultinomialNB(alpha = 1)
NB.fit(train_features, train_labels)


# In[65]:


# NB Metrics

train_predictions = NB.predict(train_features)
test_predictions = NB.predict(test_features)

print("Multinoimial Naive Bayes Training Metrics:")
printMatrix(classification_report(train_predictions, train_labels, output_dict=True))
print("")
print("Multinoimial Naive Bayes Testing Metrics:")
printMatrix(classification_report(test_predictions, test_labels, output_dict=True))
```