```
import pandas
import re

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn import metrics
from sklearn.metrics import precision_recall_fscore_support
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB

from bs4 import BeautifulSoup

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

```
True
```

```
# Dataset:
https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Jewelry_v1_00.tsv.gz
```

Read Data

```
data =
pandas.read_csv('https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Jewelry_v1_00.tsv.gz', sep='\t', on_bad_lines='skip')
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/
interactiveshell.py:3326: DtypeWarning: Columns (7) have mixed
types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Keep Reviews and Ratings

```
#Remove null value rows and reset index
data = data.dropna()
data = data.reset_index(drop=True)

#Keep only review_body column and corresponding star_rating column
data = data[['review_body', 'star_rating']]

#Removing all non-integer star_rating
data['star_rating'] = data['star_rating'].astype(int)
```

We select 20000 reviews randomly from each rating class.

```
sample_size = 20000
dataset = pandas.DataFrame()
for i in data.star_rating.unique():
    X = data[data.star_rating == i].sample(sample_size)
    dataset = dataset.append(X)
```

Data Cleaning

Pre-processing

```
X, Y = dataset['review_body'].tolist(),
dataset['star_rating'].tolist()

#Print the average character length of the reviews before cleaning
character_length_bf_cl = 0
for i in range(len(X)):
  character_length_bf_cl += len(X[i])
print('Average character length before cleaning: ',
character_length_bf_cl/len(X))

#Convert reviews to lower case
X = list(map(lambda x: str(x).lower(), X))

#Remove HTML and URLs from reviews
X = list(map(lambda x: re.sub('<.*>', '', x), X))
X = list(map(lambda x: re.sub(r'https?://\S+', '', x), X))

#Remove non-alphabetical characters
X = list(map(lambda x: re.sub('[^a-z ]', '', x), X))

#Remove extra spaces
X = list(map(lambda x: re.sub(' +', ' ', x), X))

#Expand contractions
contractions = {
```

```
"ain't": "am not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he would",
"he'd've": "he would have",
"he'll": "he will",
"he'll've": "he will have",
"he's": "he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "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",
"I've": "I have",
"isn't": "is not",
"it'd": "it would",
"it'd've": "it would have",
"it'll": "it will",
"it'll've": "it will have",
"it's": "it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
```

```
"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",
"so've": "so have",
"so's": "so is",
"that'd": "that would",
"that'd've": "that would have",
"that's": "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 is",
"what've": "what have",
"when's": "when is",
"when've": "when have",
"where'd": "where did",
"where's": "where is",
"where've": "where have",
"who'll": "who will",
"who'll've": "who will have",
"who's": "who is",
"who've": "who have",
"why's": "why is",
"why've": "why have",
```

```
"will've": "will have",
"won't": "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"
}

def decontraction(s):
    for word in s.split(' '):
        if word in contractions.keys():
            s = re.sub(word, contractions[word], s)
    return s
X = list(map(decontraction, X))

#Print the average character length of the reviews after cleaning
character_length_af_cl = 0
for i in range(len(X)):
  character_length_af_cl += len(X[i])
print('Average character length after cleaning: ',
character_length_af_cl/len(X))
```

```
Average character length before cleaning:  189.45369
Average character length after cleaning:  172.96123
```

remove the stop words

```
#Print the average character length of the reviews before pre-
processing
character_length_bf_pp = 0
for i in range(len(X)):
  character_length_bf_pp += len(X[i])
print('Average character length before pre-processing: ',
character_length_bf_pp/len(X))

# remove stop words
stopWords =set(stopwords.words('english'))
def rmstopWords(s):
    wordlist = s.split(' ')
    newlist = []
```

```
    for word in wordlist:
        if word not in stopWords:
            newlist.append(word)
    s = ' '.join(newlist)
    return s

X = list(map(rmstopWords, X))
```

```
Average character length before pre-processing:  172.96123
```

perform lemmatization

```
# perform lemmatization
wnl = WordNetLemmatizer()
X = list(map(lambda x: ' '.join(map(wnl.lemmatize, x.split(' '))), X))

#Print the average character length of the reviews after pre-
processing
character_length_af_pp = 0
for i in range(len(X)):
  character_length_af_pp += len(X[i])
print('Average character length after pre-processing: ',
character_length_af_pp/len(X))
```

```
Average character length after pre-processing:  105.73715
```

TF-IDF Feature Extraction

```
vectorizer = TfidfVectorizer()
tfidf = vectorizer.fit_transform(X)

#Splitting data into training and testing set
X_train, X_test, Y_train, Y_test = train_test_split(tfidf, Y,
test_size=0.2, random_state=1)
```

Perceptron

```
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)

Y_test_predict = perceptron.predict(X_test)

report = metrics.classification_report(Y_test, Y_test_predict,
output_dict = True)
#print(report)
print(report['1']['precision'], ',', report['1']['recall'], ',',
report['1']['f1-score'], '\n')
print(report['2']['precision'], ',', report['2']['recall'], ',',
report['2']['f1-score'], '\n')
print(report['3']['precision'], ',', report['3']['recall'], ',',
```

```
report['3']['f1-score'], '\n')
print(report['4']['precision'], ',', report['4']['recall'], ',',
report['4']['f1-score'], '\n')
print(report['5']['precision'], ',', report['5']['recall'], ',',
report['5']['f1-score'], '\n')
print(report['weighted avg']['precision'], ',', report['weighted avg']
['recall'], ',', report['weighted avg']['f1-score'], '\n')
```

```
0.5165441176470589 , 0.48821047406304297 , 0.5019777976266429

0.2964793082149475 , 0.24552429667519182 , 0.26860660324566316

0.3016611295681063 , 0.33922789539227893 , 0.3193434935521688

0.3430073126142596 , 0.378562421185372 , 0.3599088838268792

0.5761752399704652 , 0.5736339132565548 , 0.5749017681728881

0.40814774645856994 , 0.40655 , 0.4064048666113688
```

SVM

```
svm = LinearSVC()
svm.fit(X_train, Y_train)

Y_test_predict = svm.predict(X_test)

report = metrics.classification_report(Y_test, Y_test_predict,
output_dict = True)
#print(report)
print(report['1']['precision'], ',', report['1']['recall'], ',',
report['1']['f1-score'], '\n')
print(report['2']['precision'], ',', report['2']['recall'], ',',
report['2']['f1-score'], '\n')
print(report['3']['precision'], ',', report['3']['recall'], ',',
report['3']['f1-score'], '\n')
print(report['4']['precision'], ',', report['4']['recall'], ',',
report['4']['f1-score'], '\n')
print(report['5']['precision'], ',', report['5']['recall'], ',',
report['5']['f1-score'], '\n')
print(report['weighted avg']['precision'], ',', report['weighted avg']
['recall'], ',', report['weighted avg']['f1-score'], '\n')
```

```
0.554019014693172 , 0.6363862000496401 , 0.5923530091255631

0.3709090909090909 , 0.3391304347826087 , 0.35430861723446894

0.3991017964071856 , 0.3320049813200498 , 0.36247450713800133
```

```
0.4367816091954023 , 0.40252206809583857 , 0.4189526184538654

0.6121174266083698 , 0.7204116638078902 , 0.6618640252138677

0.4757340583338357 , 0.48795 , 0.47947431661900564
```

Logistic Regression

```
logistic = LogisticRegression(solver = 'saga')
logistic.fit(X_train, Y_train)

Y_test_predict = logistic.predict(X_test)

report = metrics.classification_report(Y_test, Y_test_predict,
output_dict = True)
#print(report)
print(report['1']['precision'], ',', report['1']['recall'], ',',
report['1']['f1-score'], '\n')
print(report['2']['precision'], ',', report['2']['recall'], ',',
report['2']['f1-score'], '\n')
print(report['3']['precision'], ',', report['3']['recall'], ',',
report['3']['f1-score'], '\n')
print(report['4']['precision'], ',', report['4']['recall'], ',',
report['4']['f1-score'], '\n')
print(report['5']['precision'], ',', report['5']['recall'], ',',
report['5']['f1-score'], '\n')
print(report['weighted avg']['precision'], ',', report['weighted avg']
['recall'], ',', report['weighted avg']['f1-score'], '\n')
```

```
0.5929531757070005 , 0.634896996773393 , 0.613208677933597

0.3906331763474621 , 0.38184143222506395 , 0.3861872736678738

0.42608222161720666 , 0.38978829389788294 , 0.4071279916753382

0.46931696905016007 , 0.4436317780580076 , 0.45611305587968365

0.6531622777402656 , 0.7111002205341828 , 0.680900985452839

0.5076750610988537 , 0.51385 , 0.5101237039104157
```

Naive Bayes

```
mnb = MultinomialNB()
mnb.fit(X_train, Y_train)
```

```
Y_test_predict = mnb.predict(X_test)

Y_test_predict = logistic.predict(X_test)

report = metrics.classification_report(Y_test, Y_test_predict,
output_dict = True)
#print(report)
print(report['1']['precision'], ',', report['1']['recall'], ',',
report['1']['f1-score'], '\n')
print(report['2']['precision'], ',', report['2']['recall'], ',',
report['2']['f1-score'], '\n')
print(report['3']['precision'], ',', report['3']['recall'], ',',
report['3']['f1-score'], '\n')
print(report['4']['precision'], ',', report['4']['recall'], ',',
report['4']['f1-score'], '\n')
print(report['5']['precision'], ',', report['5']['recall'], ',',
report['5']['f1-score'], '\n')
print(report['weighted avg']['precision'], ',', report['weighted avg']
['recall'], ',', report['weighted avg']['f1-score'], '\n')
```

```
0.5929531757070005 , 0.634896996773393 , 0.613208677933597

0.3906331763474621 , 0.38184143222506395 , 0.3861872736678738

0.42608222161720666 , 0.38978829389788294 , 0.4071279916753382

0.46931696905016007 , 0.4436317780580076 , 0.45611305587968365

0.6531622777402656 , 0.7111002205341828 , 0.680900985452839

0.5076750610988537 , 0.51385 , 0.5101237039104157
```