January 23, 2023

```
[ ]: ! pip install bs4
     ! pip install contractions
     ! pip install scikit-learn
     ! pip install pandas
     ! pip install numpy
     # Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/
      ↪amazon_reviews_us_Beauty_v1_00.tsv.gz
```

```
[ ]: import pandas as pd
     import numpy as np
     import nltk
     nltk.download('wordnet')
     nltk.download('punkt')
     import re
     from bs4 import BeautifulSoup
     import contractions
     from nltk.corpus import wordnet
     from sklearn.feature_extraction.text import TfidfVectorizer
```

0.1 Read Data

```
[ ]: data = pd.read_csv('amazon_reviews_us_Beauty_v1_00.tsv', sep='\t',␣
      ↪encoding='utf-8', on_bad_lines='skip')
```

```
[ ]: data['star_rating'] = pd.to_numeric(data['star_rating'],errors='coerce')␣
      ↪#results in NaNs
```

0.2 Keep Reviews and Ratings

```
[ ]: df = data[['review_body','star_rating']] #dropping other columns

     df = df.dropna(thresh=2)
```

We form three classes and select 20000 reviews randomly from each class.

```
[ ]: class_a = df.loc[df['star_rating'].isin([1,2])].sample(n=20000, random_state=1)
     class_b = df.loc[df['star_rating'].isin([3])].sample(n=20000,random_state=2)
     class_c = df.loc[df['star_rating'].isin([4,5])].sample(n=20000,random_state=3)
```

```
df_sampled = pd.concat([class_a, class_b, class_c])
df_sampled['star_rating'].value_counts()
```

[ ]:
```
3.0    20000
5.0    16344
1.0    12553
2.0     7447
4.0     3656
Name: star_rating, dtype: int64
```

[ ]:
```
df_sampled['star_rating'] = df_sampled['star_rating'].replace([1,2],"A")
df_sampled['star_rating'] = df_sampled['star_rating'].replace([3],"B")
df_sampled['star_rating'] = df_sampled['star_rating'].replace([4,5],"C")
df_sampled['star_rating'].value_counts()
#df_sampled
```

[ ]:
```
A    20000
B    20000
C    20000
Name: star_rating, dtype: int64
```

So far, I have read the dataset from the tsv file. I converted the data type of star_rating field to int in order to enforce uniformity as there were some float and date values in them. After dropping null values, sampled 20000 reviews for each class label and merged them into a single dataframe.

1 Data Cleaning

[ ]:
```
before_clean = np.mean(df_sampled['review_body'].apply(lambda x: len(str(x))))
```

[ ]:
```
def remove_HTML(s):
    return re.sub(r'<.*?>',' ',s)

def remove_URL(s):
    return re.sub(r'https?:\/\/.*\/\w*',' ',s)

def remove_nonalphabets(s):
    return re.sub(r'[^a-zA-Z]',' ',s)

def remove_multispace(s):
    return re.sub(r'\s+',' ',s)
```

[ ]:
```
df_sampled['review_body'] = df_sampled['review_body'].apply(lambda x: str(x).
 ↪lower())
df_sampled['review_body'] = df_sampled['review_body'].apply(lambda x:
 ↪remove_HTML(x))
```

```
df_sampled['review_body'] = df_sampled['review_body'].apply(lambda x:
 ↪remove_URL(x))
df_sampled['review_body'] = df_sampled['review_body'].apply(lambda x:
 ↪contractions.fix(x))
df_sampled['review_body'] = df_sampled['review_body'].apply(lambda x:
 ↪remove_nonalphabets(x))
df_sampled['review_body'] = df_sampled['review_body'].apply(lambda x:
 ↪remove_multispace(x))
#df_sampled
```

[ ]:
```
after_clean = np.mean(df_sampled['review_body'].apply(lambda x: len(x)))
print("Average document lengths before and after cleaning : ",before_clean,",␣
 ↪",after_clean)
```

```
Average document lengths before and after cleaning :  269.11678333333333 ,
259.8142
```

Performed data-cleaning on the sampled dataset by performing a set of steps in particular order. 1. Remove HTML tags. 2. Remove URL links. 3. Perform standardization of contractions such as don't, won't using the contractions library. Performed standardization first in order to avoid removing the apostrophe which happens in step 4. 4. Remove non-alphabetic characters. 5. Finally, remove multiple spaces.

2 Pre-processing

[ ]:
```
df_sampled['tokenized'] = df_sampled['review_body'].apply(lambda x: nltk.
 ↪word_tokenize(x))
```

2.0.1 Remove the stop words

[ ]:
```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
df_sampled['removed_stopwords'] = df_sampled['tokenized'].apply(lambda x: [w␣
 ↪for w in x if w not in stop_words])
```

2.0.2 Perform POS Tagging and Lemmatization

[ ]:
```
df_sampled['tokenized'] = df_sampled['tokenized'].apply(lambda x: nltk.
 ↪pos_tag(x))
df_sampled['removed_stopwords'] = df_sampled['removed_stopwords'].apply(lambda␣
 ↪x: nltk.pos_tag(x))
```

[ ]:
```
#to convert nltk treebank pos tag to wordnet pos tag for lemmatization
def get_wordnet_pos_from_tag(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
```

```
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

[ ]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
df_sampled['lemmatized'] = df_sampled['tokenized'].apply(lambda x: [lemmatizer.
 ↪lemmatize(w[0],pos=get_wordnet_pos_from_tag(w[1])) for w in x])
df_sampled['lemmatized_no_stop'] = df_sampled['removed_stopwords'].apply(lambda␣
 ↪x: [lemmatizer.lemmatize(w[0],pos=get_wordnet_pos_from_tag(w[1])) for w in␣
 ↪x])
```

[ ]:
```
df_sampled['lemmatized'] = df_sampled['lemmatized'].apply(lambda x: ' '.join(x))
df_sampled['lemmatized_no_stop'] = df_sampled['lemmatized_no_stop'].
 ↪apply(lambda x: ' '.join(x))
#df_sampled
```

[ ]:
```
after_preprocessing = np.mean(df_sampled['lemmatized'].apply(lambda x : len(x)))
after_preprocessing1 = np.mean(df_sampled['lemmatized_no_stop'].apply(lambda x :
 ↪ len(x)))
print("Average document lengths before and after lemmatize without removing␣
 ↪stop words : ",after_clean,", ",after_preprocessing)
print("Average document lengths before and after lemmatize after removing stop␣
 ↪words: ",after_clean,", ",after_preprocessing1)
```

```
Average document lengths before and after lemmatize without removing stop words
:  259.8142 ,  249.02083333333334
Average document lengths before and after lemmatize after removing stop words:
259.8142 ,  149.82061666666667
```

3 Experimenting with Stemmer

[ ]:
```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
df_sampled['stemmed'] = df_sampled['tokenized'].apply(lambda x: [stemmer.
 ↪stem(w[0]) for w in x])
df_sampled['stemmed_no_stop'] = df_sampled['removed_stopwords'].apply(lambda x:␣
 ↪[stemmer.stem(w[0]) for w in x])
```

[ ]:
```
df_sampled['stemmed'] = df_sampled['stemmed'].apply(lambda x: ' '.join(x))
df_sampled['stemmed_no_stop'] = df_sampled['stemmed_no_stop'].apply(lambda x: '␣
 ↪'.join(x))
```

```
#df_sampled
```

[ ]:
```
after_preprocessing = np.mean(df_sampled['stemmed'].apply(lambda x : len(x)))
after_preprocessing1 = np.mean(df_sampled['stemmed_no_stop'].apply(lambda x :␣
↪len(x)))
print("Average document lengths before and after stemming without stop words:␣
↪",after_clean,", ",after_preprocessing)
print("Average document lengths before and after stemming after stop words :␣
↪",after_clean,", ",after_preprocessing1)
```

```
Average document lengths before and after stemming without stop words:  259.8142
,  240.39201666666668
Average document lengths before and after stemming after stop words :  259.8142
,  141.43468333333334
```

Data pre-processing was done in 4 ways: 1. Lemmatizing without removing stop words. 2. Lemmatizing after removing stop words. 3. Stemming without removing stop words. 4. Stemming after removing stop words.

[ ]:
```
df_sampled
```

[ ]:
```
                                               review_body star_rating  \
3957522  i used this only for my eyebrows as over time …           A
2728078  smelled more like pine sol not happy with this…           A
1437220           i used it and did not see any changes at           A
372183   i have used this on my body no problems then i…           A
774226   did not have safety seal on it could have just…           A
…                                                      …         …
974370   i have used this brand before and it works gre…           C
4852419  i just received my groom mate and used it on m…           C
3164809  have been using many creams for my eczema over…           C
2464573                                            love it           C
2324756                                                 ok           C

                                                 tokenized  \
3957522  [(i, NN), (used, VBD), (this, DT), (only, JJ),…
2728078  [(smelled, VBD), (more, RBR), (like, IN), (pin…
1437220  [(i, NN), (used, VBD), (it, PRP), (and, CC), (…
372183   [(i, NNS), (have, VBP), (used, VBN), (this, DT…
774226   [(did, VBD), (not, RB), (have, VB), (safety, N…
…                                                      …
974370   [(i, NNS), (have, VBP), (used, VBN), (this, DT…
4852419  [(i, NN), (just, RB), (received, VBN), (my, PR…
3164809  [(have, VBP), (been, VBN), (using, VBG), (many…
2464573                          [(love, VB), (it, PRP)]
2324756                                       [(ok, NN)]

                                         removed_stopwords  \
```

```
3957522  [(used, VBN), (eyebrows, NNS), (time, NN), (co…
2728078  [(smelled, VBN), (like, IN), (pine, NN), (sol,…
1437220          [(used, VBN), (see, NN), (changes, NNS)]
372183   [(used, VBN), (body, NN), (problems, NNS), (us…
774226   [(safety, NN), (seal, NN), (could, MD), (soap,…
…                                                   …
974370   [(used, VBN), (brand, NN), (works, VBZ), (grea…
4852419  [(received, VBN), (groom, NN), (mate, NN), (us…
3164809  [(using, VBG), (many, JJ), (creams, NNS), (ecz…
2464573                                    [(love, NN)]
2324756                                      [(ok, NN)]

                                               lemmatized  \
3957522  i use this only for my eyebrow a over time and…
2728078  smell more like pine sol not happy with this p…
1437220              i use it and do not see any change at
372183   i have use this on my body no problem then i u…
774226   do not have safety seal on it could have just …
…                                                   …
974370   i have use this brand before and it work great…
4852419  i just receive my groom mate and use it on my …
3164809  have be use many cream for my eczema over the …
2464573                                         love it
2324756                                              ok

                                       lemmatized_no_stop  \
3957522  use eyebrow time constant abuse brows leave li…
2728078                 smell like pine sol happy product
1437220                                    use see change
372183   use body problem use face frequent basis week …
774226             safety seal could soap water oil trust
…                                                   …
974370   use brand work great really love bright color …
4852419  receive groom mate use nose worked great batte…
3164809  use many cream eczema year moderate thing work…
2464573                                            love
2324756                                              ok

                                                  stemmed  \
3957522  i use thi onli for my eyebrow as over time and…
2728078  smell more like pine sol not happi with thi pr…
1437220             i use it and did not see ani chang at
372183   i have use thi on my bodi no problem then i us…
774226   did not have safeti seal on it could have just…
…                                                   …
974370   i have use thi brand befor and it work great i…
4852419  i just receiv my groom mate and use it on my n…
```

```
3164809  have been use mani cream for my eczema over th…
2464573                                          love it
2324756                                               ok

                                          stemmed_no_stop
3957522  use eyebrow time constant abus brow left littl…
2728078                 smell like pine sol happi product
1437220                                     use see chang
372183   use bodi problem use face frequent basi week e…
774226             safeti seal could soap water oil trust
…                                                     …
974370   use brand work great realli love bright color …
4852419  receiv groom mate use nose work great batteri …
3164809  use mani cream eczema year moder thing work st…
2464573                                             love
2324756                                               ok

[60000 rows x 8 columns]
```

4 TF-IDF Feature Extraction

[ ]:
```python
lemmaVectorizer = TfidfVectorizer()
lemmatfidf = lemmaVectorizer.fit_transform(df_sampled['lemmatized'])

lemmaNoStopVectorizer = TfidfVectorizer()
lemmaNoStoptfidf = lemmaNoStopVectorizer.
 ↪fit_transform(df_sampled['lemmatized_no_stop'])

stemVectorizer = TfidfVectorizer()
stemtfidf = stemVectorizer.fit_transform(df_sampled['stemmed'])

stemNoStopVectorizer = TfidfVectorizer()
stemNoStoptfidf = stemNoStopVectorizer.
 ↪fit_transform(df_sampled['stemmed_no_stop'])
```

[ ]:
```python
lemma_df = pd.DataFrame(lemmatfidf[0].T.todense(),
index = lemmaVectorizer.get_feature_names_out(), columns=["TF-IDF"])
lemma_df = lemma_df.sort_values('TF-IDF', ascending=False)
# lemma_df
```

[ ]:
```python
lemmaNoStop_df = pd.DataFrame(lemmaNoStoptfidf[0].T.todense(),
index = lemmaNoStopVectorizer.get_feature_names_out(), columns=["TF-IDF"])
lemmaNoStop_df = lemmaNoStop_df.sort_values('TF-IDF', ascending=False)
# lemmaNoStop_df
```

```
[ ]: stem_df = pd.DataFrame(stemtfidf[0].T.todense(),
     index = stemVectorizer.get_feature_names_out(), columns=["TF-IDF"])
     stem_df = stem_df.sort_values('TF-IDF', ascending=False)
     # stem_df
```

```
[ ]: stemNoStop_df = pd.DataFrame(stemNoStoptfidf[0].T.todense(),
     index = stemNoStopVectorizer.get_feature_names_out(), columns=["TF-IDF"])
     stemNoStop_df = stemNoStop_df.sort_values('TF-IDF', ascending=False)
     # stemNoStop_df
```

Following the methods followed in pre-processing, performed tf-idf scoring experiments on each method individually.

5 Training-test split

```
[ ]: from sklearn.model_selection import train_test_split
     X1_train, X1_test, y1_train, y1_test  = train_test_split(lemmatfidf,
                                  df_sampled['star_rating'],
                                  stratify=df_sampled['star_rating'],
                                  test_size=0.2, random_state=1)
```

```
[ ]: X2_train, X2_test, y2_train, y2_test  = train_test_split(lemmaNoStoptfidf,
                                  df_sampled['star_rating'],
                                  stratify=df_sampled['star_rating'],
                                  test_size=0.2, random_state=1)
```

```
[ ]: X3_train, X3_test, y3_train, y3_test  = train_test_split(stemtfidf,
                                  df_sampled['star_rating'],
                                  stratify=df_sampled['star_rating'],
                                  test_size=0.2, random_state=1)
```

```
[ ]: X4_train, X4_test, y4_train, y4_test  = train_test_split(stemNoStoptfidf,
                                  df_sampled['star_rating'],
                                  stratify=df_sampled['star_rating'],
                                  test_size=0.2, random_state=1)
```

Split the sampled dataset into training and test datasets using stratified split. The split is 80% training, 20% testing. 1. The index 1 represents Lemmatized features without removing stop words. 2. The index 2 represents Lemmatized features after removing stop words. 3. The index 3 represents Stemmed features without removing stop words. 4. The index 4 represents Stemmed features after removing stop words.

6 Perceptron

[ ]:
```
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report

p1 = Perceptron(random_state=7)
p1.fit(X1_train, y1_train)

scores = classification_report(y1_test, p1.predict(X1_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6208530805687204 ,  0.655 ,  0.637469586374696
0.539924973204716 ,  0.50375 ,  0.5212105535437144
0.6951581027667985 ,  0.7035 ,  0.6993041749502983
0.6186453855134116 ,  0.62075 ,  0.6193281049562362
```

[ ]:
```
p2 = Perceptron(random_state=7)
p2.fit(X2_train, y2_train)

scores = classification_report(y2_test, p2.predict(X2_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.5755575785161584 ,  0.63225 ,  0.6025732666190136
0.5176369371952968 ,  0.45125 ,  0.48216909309469747
0.6554989075018208 ,  0.675 ,  0.6651065402143121
0.5828978077377586 ,  0.5861666666666666 ,  0.5832829666426744
```

[ ]:
```
p3 = Perceptron(random_state=7)
p3.fit(X3_train, y3_train)

scores = classification_report(y3_test, p3.predict(X3_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
```

```
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6367866764633847 ,  0.65 ,  0.6433254979586788
0.5463142580019399 ,  0.56325 ,  0.5546528803545051
0.7413656736092803 ,  0.703 ,  0.7216732965481842
0.6414888693582016 ,  0.63875 ,  0.639883891620456
```

[ ]:
```
p4 = Perceptron(random_state=7)
p4.fit(X4_train, y4_train)

scores = classification_report(y4_test, p4.predict(X4_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6047992164544564 ,  0.6175 ,  0.6110836219693221
0.5150226003722415 ,  0.48425 ,  0.4991624790619766
0.6649819494584838 ,  0.69075 ,  0.6776210913549968
0.5949345887617272 ,  0.5975 ,  0.5959557307954317
```

7 SVM

[ ]:
```
from sklearn.svm import LinearSVC
s1 = LinearSVC(random_state=7, tol= 1e-5)
s1.fit(X1_train, y1_train)

scores = classification_report(y1_test, s1.predict(X1_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6971442885771543 ,  0.69575 ,  0.6964464464464465
0.6118943860548577 ,  0.59675 ,  0.6042273129983545
```

```
0.7684441197954711 ,  0.789 ,  0.7785864068089305
0.692494264809161 ,  0.6938333333333333 ,  0.6930867220845771
```

[ ]:
```
s2 = LinearSVC(random_state=7, tol= 1e-5)
s2.fit(X2_train, y2_train)

scores = classification_report(y2_test, s2.predict(X2_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6555364596120795 ,  0.6675 ,  0.6614641397250093
0.5720708807193864 ,  0.54075 ,  0.5559696697082637
0.7214182344428365 ,  0.74775 ,  0.734348146329487
0.649675191591434 ,  0.652 ,  0.6505939852542533
```

[ ]:
```
s3 = LinearSVC(random_state=7, tol= 1e-5)
s3.fit(X3_train, y3_train)
scores = classification_report(y3_test, s3.predict(X3_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6967437235893612 ,  0.70075 ,  0.6987411192820641
0.6148282097649186 ,  0.595 ,  0.6047516198704103
0.7722844617632733 ,  0.79275 ,  0.7823834196891192
0.6946187983725176 ,  0.6961666666666667 ,  0.6952920529471979
```

[ ]:
```
s4 = LinearSVC(random_state=7, tol= 1e-5)
s4.fit(X4_train, y4_train)

scores = classification_report(y4_test, s4.predict(X4_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
```

```
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6656826568265682 ,  0.6765 ,  0.6710477371357718
0.5816435432230523 ,  0.545 ,  0.5627258647392875
0.7231908287556723 ,  0.757 ,  0.7397092952241358
0.6568390096017643 ,  0.6595 ,  0.6578276323663983
```

8 Logistic Regression

```
[ ]: from sklearn.linear_model import LogisticRegression
lr1 = LogisticRegression(random_state=7, max_iter=300)
lr1.fit(X1_train, y1_train)

scores = classification_report(y1_test, lr1.predict(X1_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.7100903614457831 ,  0.70725 ,  0.7086673346693386
0.6207411091768217 ,  0.624 ,  0.6223662884927066
0.788235294117647 ,  0.78725 ,  0.7877423389618511
0.7063555882467505 ,  0.7061666666666667 ,  0.7062586540412988
```

```
[ ]: lr2 = LogisticRegression(random_state=7, max_iter=300)
lr2.fit(X2_train, y2_train)

scores = classification_report(y2_test, lr2.predict(X2_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6734895191122071 ,  0.68275 ,  0.6780881440099318
0.5886830753615834 ,  0.58 ,  0.584309280946984
0.748001998001998 ,  0.74875 ,  0.7483758120939529
0.6700581974919294 ,  0.6705 ,  0.6702577456836228
```

```
[ ]: lr3 = LogisticRegression(random_state=7, max_iter=300)
lr3.fit(X3_train, y3_train)

scores = classification_report(y3_test, lr3.predict(X3_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.7150389153904092 ,  0.712 ,  0.7135162219716897
0.625748502994012 ,  0.627 ,  0.6263736263736264
0.7937141431778498 ,  0.7955 ,  0.794606068173305
0.7115005205207569 ,  0.7115 ,  0.7114986388395405
```

```
[ ]: lr4 = LogisticRegression(random_state=7, max_iter=300)
lr4.fit(X4_train, y4_train)

scores = classification_report(y4_test, lr4.predict(X4_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6789149198520346 ,  0.68825 ,  0.6835505896958411
0.5915601023017902 ,  0.57825 ,  0.5848293299620734
0.748451053283767 ,  0.755 ,  0.7517112632233977
0.6729753584791973 ,  0.6738333333333333 ,  0.6733637276271041
```

9 Naive Bayes

```
[ ]: from sklearn.naive_bayes import MultinomialNB
nb1 = MultinomialNB()
nb1.fit(X1_train, y1_train)

scores = classification_report(y1_test, nb1.predict(X1_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
```

```
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6824395373291272 ,  0.649 ,  0.6652998462327011
0.5696119272089454 ,  0.6495 ,  0.6069384417708211
0.7845942228335626 ,  0.713 ,  0.7470857891290111
0.678881895790545 ,  0.6705 ,  0.6731080257108445
```

[ ]:
```
nb2 = MultinomialNB()
nb2.fit(X2_train, y2_train)

scores = classification_report(y2_test, nb2.predict(X2_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6724879097259537 ,  0.62575 ,  0.6482776482776483
0.5639668660837551 ,  0.61275 ,  0.5873472322070452
0.7337232960325534 ,  0.72125 ,  0.7274331820474027
0.6567260239474206 ,  0.65325 ,  0.6543526875106987
```

[ ]:
```
nb3 = MultinomialNB()
nb3.fit(X3_train, y3_train)

scores = classification_report(y3_test, nb3.predict(X3_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6925119490175252 ,  0.652 ,  0.6716456348184393
0.5758831225468818 ,  0.66025 ,  0.6151875145585838
0.7889254385964912 ,  0.7195 ,  0.7526150627615064
0.6857735033869661 ,  0.67725 ,  0.6798160707128431
```

[ ]:
```
nb4 = MultinomialNB()
nb4.fit(X4_train, y4_train)
```

```
scores = classification_report(y4_test, nb4.predict(X4_test),output_dict=True)
print(scores['A']['precision'],", ",scores['A']['recall'],",␣
↪",scores['A']['f1-score'])
print(scores['B']['precision'],", ",scores['B']['recall'],",␣
↪",scores['B']['f1-score'])
print(scores['C']['precision'],", ",scores['C']['recall'],",␣
↪",scores['C']['f1-score'])
print(scores['weighted avg']['precision'],", ",scores['weighted␣
↪avg']['recall'],", ",scores['weighted avg']['f1-score'])
```

```
0.6764864864864865 ,  0.62575 ,  0.6501298701298702
0.5647656430385593 ,  0.6115 ,  0.5872044172368264
0.7306626354245402 ,  0.725 ,  0.7278203036767474
0.657304921649862 ,  0.6540833333333333 ,  0.6550515303478147
```

Performed experiments on each of the features prepared using the methods described above.
Observed that stemming without removing stop words results in highest accuracy in all 4 models.