HW 1 - Report

Kevin Lee
2995359247

Data Cleaning & Formatting
To prepare the data for the models, I've first cleaned the Amazon reviews dataset. This meant I got rid of URL, HTML, any contractions and any non alphabets in that order. I had to get rid of URL & HTML first and then non-alphabets as doing so would make some of the URLs or HTMLs indistinguishable from regular words. I've utilized regex to get rid of any HTML, and BeautifulSoup4 to get rid of any URLs. After that, I've utilized regex again to drop any non-alphabet and made everything into a lowercase. Doing this has dropped the average character count of the review_body from 314.95 to 303.85.

I've also combined review_headline with review_body. Review_headline before cleaning had on average 23.72 characters, and review_body on average had 290.22 characters. Combining both headline and body gave 314.95 average characters. Adding in review_headline was good for the model, as there are some key words that have been a useful feature to determine for class, such as "Five Stars" or "Amazing". So I've combined the two columns and cleaned them up, giving me 303.85 characters.

To pre-process the review column, I got rid of all the stop words and lemmatized all the words. I also stemmed the review column to see if there was any difference between lemmatization & stemming, but found out lemmatization got me a better result by very small margin. I also utilized pos_tag, as I thought focusing on adjectives/adverbs/verbs/noun would be useful. This was not the case, and gave me a worse result (~0.8 less F1 score for some of the models). Pre-processing the review column shrank down the average character count from 303.85 to 187.54. Without the review_headline, the average character count was about 150..

To vectorize my formatted data, I used TfidfVectorizer. I also experimented with ngram_range of the TfidfVectorizer to utilize unigram/bigram/trigram as features. I tried unigram only, unigram & bigram, bigram only, unigram & bigram & trigram, bigram & trigram, and even 1-gram to 4-gram. The last combination was excessive and gave me a worse result as well as extremely extending the time to fit the models.

The dataset was split training to testing as 80 to 20. The dataset was also shuffled using a constant random_state of 42. This constant random seed was needed as my models also utilized random_state in order to keep consistent scoring across multiple sessions.

Models w/ <u>LOWER</u> scores - Stop Words

Perceptron
To get my model, I've initially used GridSearchCV to hyper-parameterize my model. The hyperparameters I explored were eta0 and max_iter. At the end, there was only 0.001 f1_score difference, no matter what combinations I've used.

```
PS C:\Users\jaehw\OneDrive\Desktop\CSCI544\HW1> python .\hw1.py
Mean Accuracy: 0.749
Config: {'eta0': 0.1}
Accuracy 0.7482847222222221 with parameter {'eta0': 0.0001}
Accuracy 0.7477569444444444 with parameter {'eta0': 0.001}
Accuracy 0.7485069444444444 with parameter {'eta0': 0.01}
Accuracy 0.7487013888888892 with parameter {'eta0': 0.1}
Accuracy 0.7484305555555555 with parameter {'eta0': 10}
```

What increased by score to greater than 0.75 was the usage of random_state. Due to the speed of Perceptron fitting, I was able to have 1000 iterations of different models with different random_state. From there, I chose the best possible random_state, 828, that got the highest average precision, recall and f1-score.

Old Score	Precision	Recall	F1-Score
Class 1	0.761	0.767	0.765
Class 2	0.690	0.672	0.681
Class 3	0.805	0.821	0.813
Average	0.752	0.753	0.753

SVM

Old Score	Precision	Recall	F1-Score
Class 1	0.765	0.787	0.776
Class 2	0.693	0.693	0.693
Class 3	0.833	0.809	0.821
Average	0.764	0.763	0.763

Logistic Regression

New Score	Precision	Recall	F1-Score
Class 1	0.789	0.787	0.776
Class 2	0.68	0.693	0.693
Class 3	0.8	0.83	0.821
Average	0.764	0.763	0.763

Naive Bayes

Naive bayes don't use random_state, so there wasn't much random tuning for this model.

Old Score	Precision	Recall	F1-Score
Class 1	0.759	0.768	0.764
Class 2	0.683	0.690	0.687
Class 3	0.829	0.809	0.819
Average	0.757	0.756	0.757

Models w/ HIGHER scores - Removing Stop Words

I initially filtered out all stopwords that nltk.corpus.stopwords contained. This allowed me to get the results above. But when trying out a version where I don't filter out all the stopwords, I got a much better result. This decreased my average char count from 303.84 to 298.65, increasing the training time but increasing the results significantly.

Perceptron

I looped over random_states to get 486 as the best result.

New Score	Precision	Recall	F1-Score
Class 1	0.789	0.803	0.796
Class 2	0.723	0.718	0.720
Class 3	0.865	0.855	0.860
Average	0.792	0.792	0.792

SVM

New Score	Precision	Recall	F1-Score
Class 1	0.789	0.822	0.805
Class 2	0.746	0.714	0.730
Class 3	0.782	0.874	0.873
Average	0.803	0.803	0.803

Logistic Regression

New Score	Precision	Recall	F1-Score
Class 1	0.790	0.815	0.801

Class 2	0.732	0.721	0.726
Class 3	0.868	0.853	0.860
Average	0.796	0.796	0.796

Naive Bayes

New Score	Precision	Recall	F1-Score
Class 1	0.785	0.795	0.790
Class 2	0.678	0.785	0.728
Class 3	0.93	0.773	0.844
Average	0.798	0.784	0.787

Python was wrote using Python3.9, and you can run the code by typing “python ./hw1.py”

hw1.py × | Untitled-2 ● | Untitled-1 9+ ●

hw1.py > ...

```python
model.fit(train_tfidf, train_result)
result = model.predict(test_tfidf)
report = classification_report(test_result, result, output_dict=True)

precision = (report["0"]["precision"] + report["1"]["precision"] + report["2"]["precision"])/3
recall = (report["0"]["recall"] + report["1"]["recall"] + report["2"]["recall"])/3
f1Score = (report["0"]["f1-score"] + report["1"]["f1-score"] + report["2"]["f1-score"])/3
print(report["0"]["precision"],",",report["0"]["recall"],",",report["0"]["f1-score"])
print(report["1"]["precision"],",",report["1"]["recall"],",",report["1"]["f1-score"])
print(report["2"]["precision"],",",report["2"]["recall"],",",report["2"]["f1-score"])
print(precision,",",recall,",",f1Score)
```

PROBLEMS 10 | OUTPUT | DEBUG CONSOLE | TERMINAL

```
KERNELBASE.dll     00007FFAE35A31A7  Unknown               Unknown  Unknown
KERNEL32.DLL       00007FFAE52C26BD  Unknown               Unknown  Unknown
Before - Review Headline Avg Character Count 23.727083333333333
Before - Review Body Avg Character Count 290.2242
Before - Total Review Avg Character Count 314.9512833333333
After - Total Review Avg Character Count 303.84655
Data Formatting
Before - Cleaned Review Avg Character Count 303.84655
After - Cleaned Review Avg Character Count 298.6517166666667
Perceptron
0.7888475836431227 , 0.8031794095382286 , 0.7959489872468117
0.7226680040120361 , 0.7179870453413054 , 0.720319920019995
0.8649735981895902 , 0.8550832711906537 , 0.8600000000000001
0.7921630619482497 , 0.7920832420233959 , 0.7920896357556023
SVM
0.7894354252483644 , 0.822104466313399 , 0.8054388133498146
0.7459656428943259 , 0.7140009965122073 , 0.7296334012219959
0.8724882163234929 , 0.8742232165050957 , 0.8733548547305687
0.8026297614887278 , 0.8034428931102341 , 0.8028090231007932
Logistic Regression
0.7894865525672372 , 0.8147867776936664 , 0.8019371662734384
0.7314791403286979 , 0.7207274539113104 , 0.726063496047183
0.8677623261694059 , 0.8530947054436987 , 0.8603660065179242
0.7962426730217803 , 0.7962029790162252 , 0.796122222946182
Multinomial Naive Bayes
0.7848953140578265 , 0.794600050466818 , 0.789717868338558
0.6782870669248978 , 0.7852516193323368 , 0.7278605241888927
0.9302604010775217 , 0.7725577926920209 , 0.844106463878327
0.7978142606867488 , 0.7841364874970586 , 0.7872282854685926
PS C:\Users\jaehw\OneDrive\Desktop\CSCI544\HW1>
```

In [1]:
```python
# from google.colab import drive
# drive.mount('/content/drive')
```

In [1]:
```python
!pip install bs4
!pip install contractions
```

```
Requirement already satisfied: bs4 in c:\users\jaehw\anaconda3\lib\site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in c:\users\jaehw\anaconda3\lib\site-packages (from bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\jaehw\anaconda3\lib\site-packages (from beautifulsoup4->bs4) (2.3.1)
Requirement already satisfied: contractions in c:\users\jaehw\anaconda3\lib\site-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in c:\users\jaehw\anaconda3\lib\site-packages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in c:\users\jaehw\anaconda3\lib\site-packages (from textsearch>=0.0.21->contractions) (0.3.1)
Requirement already satisfied: pyahocorasick in c:\users\jaehw\anaconda3\lib\site-packages (from textsearch>=0.0.21->contractions) (2.0.0)
```

In [2]:
```python
import pandas as pd
import numpy as np
import re
import contractions
from collections import defaultdict

from bs4 import BeautifulSoup

from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn.metrics import f1_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import accuracy_score

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.corpus import wordnet as wn
from nltk import pos_tag
from nltk.tokenize import word_tokenize

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

pd.options.display.max_colwidth = 500
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\jaehw\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\jaehw\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\jaehw\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jaehw\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\jaehw\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Read Data

In [4]:
```python
base_path = ""

df = pd.read_csv(base_path + "amazon_reviews_us_Beauty_v1_00.tsv.gz", compression='gzip', header=0,sep='\t', quotechar='"', on_bad_lines='skip')
```

```
C:\Users\jaehw\AppData\Local\Temp\ipykernel_252508\1914733243.py:3: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(base_path + "amazon_reviews_us_Beauty_v1_00.tsv.gz", compression='gzip', header=0,sep='\t', quotechar='"', on_bad_lines='skip')
```

Keep Reviews and Ratings

In [5]:
```python
parsed_df = df[["star_rating", "review_headline", "review_body"]]
parsed_df.dropna()
```

Out[5]:

	star_rating	review_headline	review_body
0	5	Five Stars	Love this, excellent sun block!!
1	5	Thank you Alba Bontanica!	The great thing about this cream is that it doesn't smell weird like all those chemical laden ones. I get a nice healthy un-fake looking tan that isn't orange and it makes my skin soft too.
2	5	Five Stars	Great Product, I'm 65 years old and this is all it claims to be!
3	5	GOOD DEAL!	I use them as shower caps & conditioning caps. I like that they're in bulk. It saves a lot of money.
4	5	this soaks in quick and provides a nice base for makeup	This is my go-to daily sunblock. It leaves no white cast at all and has a clean, pleasant scent. If you're a makeup wearer, this soaks in quick and provides a nice base for makeup. I've been using this brand for over a year. With daily use, this tube will last you a couple months.
...	...	...	...
5094302	5	Great Little Grooming Tool	After watching my Dad struggle with his scissors to clip, what he affectionately calls his 'tuffs of ear hair'. I bought him this electric clippers...now we do we hear him mumble..and fuss about how he is certain he will cut off his ear someday.....This is a great invention...it moves at lightenind speed and clips those hairs neaty...... Great Price too!
5094303	3	Not bad for the price	Like most sound machines, the sounds choices are limited and most have a very noticeable cycle. The brook sound actually had a click at the end! However, the ocean and white noise had a good cycle. Unfortunately, only after a year, it broke. Now, I'm looking for a better one with more sounds.
5094304	5	Best Curling Iron Ever	I bought this product because it indicated 30 second heat up time. It is great. You plug it in, hit the on button, select a heat level (1-15) , and in less than 30 seconds it is hot. No more waiting around for the iron to heat up. Quick touch ups take no time at all. I'll never go back to the "old style" plug and wait.
5094305	5	The best electric toothbrush ever, REALLY!	We have used Oral-B products for 15 years; this new model is even better. It is stronger yet thinner; generates different vibrations (3) around the toothbrush head and varies this according to pressure. Also has a built-in timer. Enjoy!
5094306	5	Smooth and shiny teeth!	I love this toothbrush. It's easy to use, and it trains aggressive brushers (read: Type As) to treat their gums with a little more TLC. Your teeth feel cleaner longer after using a sonicare. It's almost like getting a full dental cleaning every time you brush.

5093876 rows × 3 columns

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

In [7]:
```python
class1_df = parsed_df.loc[parsed_df['star_rating'].isin([1,2])].sample(20000)
class2_df = parsed_df.loc[parsed_df['star_rating'] == 3].sample(20000)
class3_df = parsed_df.loc[parsed_df['star_rating'].isin([4,5])].sample(20000)
```

In [8]:
```python
class1_df["class"] = 1
class2_df["class"] = 2
class3_df["class"] = 3
```

In [9]:
```python
final_df = pd.concat([class1_df, class2_df, class3_df])

final_df['review_headline'] = final_df['review_headline'].apply(str)
final_df['review_body'] = final_df['review_body'].apply(str)
```

In [50]:
```python
final_df = pd.read_pickle("./finaldf.pkl") # Use this as no constant sample (forgot)
```

In [51]:
```python
final_df['review'] = final_df[['review_headline', 'review_body']].agg(' '.join, axis=1)
```

In [52]:
```python
final_df = final_df.drop('star_rating', axis=1)
final_df = final_df.drop('review_headline', axis=1)
final_df = final_df.drop('review_body', axis=1)
```

In [39]:
```python
print("Review Headline Avg Character Count",(final_df['review_headline'].str.len()).mean())
print("Review Body Avg Character Count",(final_df['review_body'].str.len()).mean())
```

```
Review Headline Avg Character Count 23.727083333333333
Review Body Avg Character Count 290.2242
```

In [40]:
```python
print("Review Avg Character Count",(final_df['review'].str.len()).mean())
```

```
Review Avg Character Count 314.9512833333333
```

Data Cleaning

Pre-processing

In [53]:
```python
# Lowercasing
final_df["review"] = final_df["review"].str.lower()
```

In [54]:
```python
# class1_df[class1_df["review_body"].str.contains("1", na=False)]
def getRidOfNonAlphabet(s):
  return re.sub(r"[^a-zA-Z]+", ' ', s)

def getRidOfHTML(s):
  return BeautifulSoup(s, "lxml").text

def getRidOfURL(s):
  return re.sub(r'http\S+', '', s)

def contractions(s):
  # Also gets rid of extra spaces
  import contractions
  ans = []
  for word in s.split():
    ans.append(contractions.fix(word))
  return ' '.join(ans)
```

In [55]:
```python
final_df["review"] = final_df["review"].apply(getRidOfURL).apply(getRidOfHTML).apply(contractions).apply(getRidOfNonAlphabet)
```

```
C:\Users\jaehw\anaconda3\lib\site-packages\bs4\__init__.py:435: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to open this file and pass the filehandle into Beautiful Soup.
  warnings.warn(
```

In [44]:
```python
print("Cleaned Review Body Avg Character Count",(final_df['review'].str.len()).mean())
```

```
Cleaned Review Body Avg Character Count 303.84655
```

remove the stop words

In [45]:
```python
def getRidOfStopWords(s):
  stopWords = set(stopwords.words('english'))
  words = word_tokenize(s)
  filteredWords = []
  for word in words:
      if word not in stopWords:
          filteredWords.append(word)
  return ' '.join(filteredWords)
```

In [46]:
```python
final_df["review"] = final_df["review"].apply(getRidOfStopWords)
```

perform lemmatization

In [56]:
```python
def lemmatize(s):
  lemmatizer = WordNetLemmatizer()
  words = word_tokenize(s)

  filteredWords = []
  for word in words:
      filteredWords.append(lemmatizer.lemmatize(word))
  return ' '.join(filteredWords)

def lemattizeWithDict(l):
  lemmatizer = WordNetLemmatizer()

  tags = defaultdict(lambda : wn.NOUN)
  tags['J'] = wn.ADJ
  tags['V'] = wn.VERB
  tags['R'] = wn.ADV

  words = []
  for word, t in pos_tag(l.split()):
      if word not in stopwords.words('english'):
          words.append(lemmatizer.lemmatize(word, tags[t[0]]))
  return str(' '.join(words))
```

In [57]:
```python
final_df["formatted_review"] = final_df["review"].apply(lemmatize)
# final_df["formatted_review_dict"] = final_df["review"].apply(lemattizeWithDict)
```

In [58]:
```python
print("Formatted Review Body Avg Character Count w/ Stop Words",(final_df['formatted_review'].str.len()).mean())
```

```
Formatted Review Body Avg Character Count w/ Stop Words 298.6517166666667
```

In [49]:
```python
print("Formatted Review Body Avg Character Count",(final_df['formatted_review'].str.len()).mean())
```

```
Formatted Review Body Avg Character Count 184.49395
```

TF-IDF Feature Extraction

In [15]:
```python
# final_df = pd.read_pickle("./final_final_df.pkl")

train_x, test_x, train_y, test_y = train_test_split(final_df['formatted_review'], final_df['class'], test_size=0.2, random_state=42)

vectorizer = TfidfVectorizer(ngram_range=(1,3))
vectorizer.fit(final_df['formatted_review'])
train_tfidf = vectorizer.transform(train_x)
test_tfidf = vectorizer.transform(test_x)

Encoder = LabelEncoder()
train_result = Encoder.fit_transform(train_y)
test_result = Encoder.fit_transform(test_y)
```

Perceptron

In [17]:
```python
perceptron = Perceptron(random_state=486)
perceptron.fit(train_tfidf, train_result)
result = perceptron.predict(test_tfidf)
report = classification_report(test_result, result)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.79      0.80      0.80      3963
           1       0.72      0.72      0.72      4014
           2       0.86      0.86      0.86      4023

    accuracy                           0.79     12000
   macro avg       0.79      0.79      0.79     12000
weighted avg       0.79      0.79      0.79     12000
```

SVM

In [24]:
```python
model = svm.LinearSVC()
model.fit(train_tfidf, train_result)
result = model.predict(test_tfidf)
report = classification_report(test_result, result)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.79      0.82      0.81      3963
           1       0.75      0.71      0.73      4014
           2       0.87      0.87      0.87      4023

    accuracy                           0.80     12000
   macro avg       0.80      0.80      0.80     12000
weighted avg       0.80      0.80      0.80     12000
```

Logistic Regression

In [27]:
```python
model = LogisticRegression()
model.fit(train_tfidf, train_result)
result = model.predict(test_tfidf)
report = classification_report(test_result, result)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      3963
           1       0.73      0.72      0.73      4014
           2       0.87      0.85      0.86      4023

    accuracy                           0.80     12000
   macro avg       0.80      0.80      0.80     12000
weighted avg       0.80      0.80      0.80     12000
```

```
C:\Users\jaehw\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Naive Bayes

In [18]:
```python
model = naive_bayes.MultinomialNB()
model.fit(train_tfidf, train_result)
result = model.predict(test_tfidf)
report = classification_report(test_result, result)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.78      0.79      0.79      3963
           1       0.68      0.79      0.73      4014
           2       0.93      0.77      0.84      4023

    accuracy                           0.78     12000
   macro avg       0.80      0.78      0.79     12000
weighted avg       0.80      0.78      0.79     12000
```