Michael Royster

CSCI 544 - Homework #1

2. Data Cleaning
During the data cleaning stage, I found that the order in which you clean the data can have an impact on the quality of your samples. For example, removing non-alpha characters before performing contractions would greatly lower the quality.

3. Pre-processing
During pre-processing it was clear that properly lemmatizing each review in the corpus made a large difference. By default, the lemmatizer will assume the part of speech for every word to be a noun. It is important to make your best effort to properly tag each word with the part of speech to acquire the appropriate lemma. Additionally, removing stop words was detrimental to all models except the perceptron.

5. Perceptron
The average precision for the trained perceptron was 0.66 and the recall was 0.65. In my experiments I found significant improvements when including 2-gram words in my tf-idf vectorizer and including more features (although there was a point of diminishing returns). I also attempted to filter out features with low tf-idf scores, but this only hurt my model. Removing or including stop words also had very low impact on the overall model.

6. SVM
The SVM model provided better precision and recall than the perceptron model. This model was similarly affected by the n-gram inclusion and number of features. However, skipping stop word removal significantly improved prediction performance.

7. Logistic Regression
The logistic regression model was the best performing model with the highest precision, recall f1-score and accuracy. It also took the longest time to train. This model was weaker in all metrics for class 2. Interestingly, the perceptron had a higher precision for class 1, but the logistic regression model was significantly better at identifying class 3. I did need to increase the maximum number of iterations as occasionally this would fail to converge. (Usually when the number of features was higher.)

8. Naive Bayes
The precision, recall and f1-scores were strong for the naive bayes model when predicting class 1 and 3. However, skipping the stop word removal provided a significant gain in precision and recall at almost 0.05 per metric.

Summary

Overall, the most important factors for creating a successful model in terms of precision were driven by the data cleaning and pre-processing. Including part of speech tagging and proper lemmatization added the largest boost. Skipping the removal of stop words also provided a significant boost.

Results when removing stop words:

```
Data Cleaning avg length (before, after): 294.8065833333333, 284.4435
Data Pre-processing avg length (before, after): 284.4435, 164.11695

Model: (Precision, Recall, F1-Score)
Perceptron Class 1: 0.729072907290729, 0.40520260130065033, 0.5209003215434083
Perceptron Class 2: 0.45258329960879534, 0.8391695847923962, 0.5880290947331522
Perceptron Class 3: 0.8320610687022901, 0.490868151113335, 0.6174665617623918
Perceptron Average: 0.6712256822135706, 0.578420745434837, 0.5754618238670866

SVM Class 1: 0.6655574043261231, 0.7003501750875438, 0.6825106642291286
SVM Class 2: 0.6036752605595173, 0.5505252626313156, 0.575876504447933
SVM Class 3: 0.7391304347826086, 0.7655741806354766, 0.7521199459260169
SVM Average: 0.6694485568280562, 0.672142082881681, 0.6701622049726659

Regression Class 1: 0.6773255813953488, 0.6993496748374187, 0.6881614570514398
Regression Class 2: 0.5948586118251928, 0.5787893946973487, 0.5867139959432048
Regression Class 3: 0.7532075471698113, 0.7490617963472604, 0.7511289513296537
Regression Average: 0.6751240699185742, 0.6757275077128325, 0.675328481575687

Naive Bayes Class 1: 0.6700389105058365, 0.6460730365182591, 0.6578377690054756
Naive Bayes Class 2: 0.5609983671565197, 0.6015507753876939, 0.5805672902836452
Naive Bayes Class 3: 0.7421448974292392, 0.715036277207906, 0.7283384301732926
Naive Bayes Average: 0.6577203527990302, 0.6542149587259235, 0.6555750965096769
```

In [205…

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('universal_tagset')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
import re
from bs4 import BeautifulSoup
```

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

In [206…

```
! pip install bs4 # in case you don't have it installed
! pip install contractions
# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Beauty_
import requests
import contractions

url = 'https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Beauty_v1_
url_data = requests.get(url)

with open("data.tsv.gz", "wb") as f:
    f.write(url_data.content)
```

```
ERROR: Invalid requirement: '#'
WARNING: You are using pip version 22.0.4; however, version 22.3.1 is available.
You should consider upgrading via the 'C:\Users\michr\AppData\Local\Microsoft\Windo
wsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip install
--upgrade pip' command.
WARNING: You are using pip version 22.0.4; however, version 22.3.1 is available.
You should consider upgrading via the 'C:\Users\michr\AppData\Local\Microsoft\Windo
wsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip install
--upgrade pip' command.
```

```
Requirement already satisfied: contractions in c:\users\michr\appdata\local\package
s\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-packages\pytho
n39\site-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in c:\users\michr\appdata\local\p
ackages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-package
s\python39\site-packages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in c:\users\michr\appdata\local\packages\py
thonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-packages\python39\
site-packages (from textsearch>=0.0.21->contractions) (0.3.1)
Requirement already satisfied: pyahocorasick in c:\users\michr\appdata\local\packag
es\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-packages\pyth
on39\site-packages (from textsearch>=0.0.21->contractions) (2.0.0)
```

In [207…

```
# import gzip

# with gzip.open('data.tsv.gz', 'rb') as z:
#     with open('data.tsv', 'wb') as f:
#         f.write(z.read())
```

Read Data

In [208…

```
df = pd.read_csv("data.tsv.gz", sep='\t', on_bad_lines="skip", compression="gzip")
# print(df.head(10))
```

```
C:\Users\michr\AppData\Local\Temp\ipykernel_19644\2767852299.py:1: DtypeWarning: Co
lumns (7) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("data.tsv.gz", sep='\t', on_bad_lines="skip", compression="gzi
p")
```

Keep Reviews and Ratings

In [209…

```
df = df[['review_body', 'star_rating']]
df.head(1)
```

Out[209]:

	review_body	star_rating
0	Love this, excellent sun block!!	5

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

In [210…
```
def categorize(x):
    if x > 3:
        return 3
    elif x < 3:
        return 1
    else:
        return 2

# Form classes
df = df[df['star_rating'].apply(lambda x: isinstance(x, int))]
df['rating_class'] = df['star_rating'].apply(lambda x: categorize(x))
df.drop(columns=['star_rating'], inplace=True)

print(df.head())
```

```
                                              review_body  rating_class
196608  This is my favorite clubman scent. Not too swe...             3
196609                                    best I ever used             3
196610                                Amazing into to tat's             3
196611  This is my second pure badger brush from Omega...             3
196612                                       pretty good :-)             3
```

In [259…
```
class_one = df.query('rating_class == 1').sample(n=20000)
class_two = df.query('rating_class == 2').sample(n=20000)
class_three = df.query('rating_class == 3').sample(n=20000)

data_set = pd.concat([class_one, class_two, class_three])
data_set.reset_index(drop=True, inplace=True)
print(data_set.head())
print(data_set.shape[0])
```

```
                                         review_body  rating_class
0  What a big mistake was buying this product jus...             1
1                     Completely not for everyday use             1
2  Nioxin has changed its formula. It sure doesn'...             1
3  light is too dim.  Head is the wrong shape.  T...             1
4  Nothing like getting something that expired 2 ...             1
60000
```

Data Cleaning

In [260…

```
def remove_html(text):
    if text == '': return text
    if not isinstance(text, str):
        return str(text)
    if len(str(text)) < 3: return text
    soup = BeautifulSoup(text, 'html.parser')
    return soup.get_text()

def clean_non_alpha(word):
    return '' if re.search('[^a-zA-Z]', word) != None else word

before_cleaning = np.sum(data_set['review_body'].str.len())/len(data_set['review_bo

# lowercase
data_set['review_body'] = data_set['review_body'].str.lower()
# remove HTML
data_set['review_body'] = data_set['review_body'].apply(lambda x: remove_html(x))
# remove URLs
data_set['review_body'] = data_set['review_body'].replace('http\S+','',regex=True)
# perform contractions
data_set['review_body'] = data_set['review_body'].apply(lambda x: contractions.fix(
# remove non-alpha characters
# data_set['review_body'].replace('[^a-zA-Z\s]',' ',regex=True, inplace=True)
data_set['review_body'] = data_set['review_body'].apply(lambda text: " ".join(clean
# remove extra spaces
data_set['review_body'] = data_set['review_body'].replace('\s+',' ',regex=True)

after_cleaning = np.sum(data_set['review_body'].str.len())/len(data_set['review_bod
print(f"Average length before and after cleaning: {before_cleaning}, {after_cleanin

# print(data_set.head())
```

```
C:\Users\michr\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\bs4\__init__.py:435: MarkupRe
semblesLocatorWarning: The input looks more like a filename than markup. You may wa
nt to open this file and pass the filehandle into Beautiful Soup.
  warnings.warn(
Average length before and after cleaning: 293.66876666666667, 277.9042
```

In [261…

```
# data_set.to_csv('temp.tsv', sep='\t', index=False)
```

Pre-processing

remove the stop words

In [262…

```
# from nltk.corpus import stopwords

before_preproc = np.sum(data_set['review_body'].str.len())/len(data_set['review_bod
# stop_words = set(stopwords.words('english'))
# data_set['review_body'] = data_set['review_body'].apply(lambda text: " ".join(wor

# data_set.head()
```

In [263…

```
# print(data_set.iloc[1,0])
```

perform lemmatization

In [264…

```
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

def get_pos(x):
    return x if x in ['n','v','a','r','s'] else 'n'

lemmatizer = WordNetLemmatizer()
data_set['review_body'] = data_set['review_body'].apply(lambda body: " ".join([lemm

after_preproc = np.sum(data_set['review_body'].str.len())/len(data_set['review_body
print(f"Average length before and after pre-processing: {before_preproc}, {after_pr
```

```
Average length before and after pre-processing: 277.9042, 266.5327
```

In [265…

```
data_set.head()
```

Out[265]:

	review_body	rating_class
0	what a big mistake be buy this product just be...	1
1	completely not for everyday use	1
2	nioxin have change it formula it sure do not w...	1
3	light be too dim head be the wrong shape too m...	1
4	nothing like get something that expire year ago	1

TF-IDF Feature Extraction

In [266…

```
from sklearn.feature_extraction.text import TfidfVectorizer

data_set['review_body'] = data_set['review_body'].replace('', np.nan)
data_set = data_set.dropna(subset=['review_body'])
data_set.reset_index(drop=True, inplace=True)

tfidf = TfidfVectorizer(max_features=2000, ngram_range=(1,2)) # try max_features, m
x = tfidf.fit_transform(data_set['review_body'])

tfidf_values = pd.DataFrame(x.toarray(), columns=tfidf.get_feature_names_out())
full_set = pd.concat([data_set, tfidf_values], axis=1)
print(full_set.shape)
```

```
(59965, 2002)
```

In [267…

```
# full_set.iloc[0:full_set.shape[0], list(range(2,full_set.shape[1]))]
full_one = full_set.query('rating_class == 1')
full_two = full_set.query('rating_class == 2')
full_three = full_set.query('rating_class == 3')

# Calculating number of training rows
one_train_size = int(full_one.shape[0] * 0.8)
two_train_size = int(full_two.shape[0] * 0.8)
three_train_size = int(full_three.shape[0] * 0.8)

# Sampling the 80% for training
train_one = full_one.iloc[0:one_train_size, list(range(0,full_one.shape[1]))]
train_two = full_two.iloc[0:two_train_size, list(range(0,full_one.shape[1]))]
train_three = full_three.iloc[0:three_train_size, list(range(0,full_one.shape[1]))]

# Sampling the 20% for testing
test_one = full_one.iloc[one_train_size:full_one.shape[0], list(range(0,full_one.sh
test_two = full_two.iloc[two_train_size:full_two.shape[0], list(range(0,full_one.sh
test_three = full_three.iloc[three_train_size:full_three.shape[0], list(range(0,ful

# Combining the training and testing population
train_population = pd.concat([train_one, train_two, train_three])
test_population = pd.concat([test_one, test_two, test_three])

# train_population.to_csv('Training.csv', index=False)
# test_population.to_csv('Testing.csv', index=False)

assert train_population.shape[0] + test_population.shape[0] == full_set.shape[0], "
```

Perceptron

In [268…

```
from sklearn.linear_model import Perceptron
from sklearn import metrics

perceptron = Perceptron()

perceptron.fit(train_population.iloc[0:train_population.shape[0], list(range(2,trai
# score = perceptron.score(test_population.iloc[0:full_set.shape[0], list(range(2,t

prediction = perceptron.predict(test_population.iloc[0:test_population.shape[0], li
perceptron_results = pd.DataFrame(zip(test_population.iloc[0:test_population.shape[
score = perceptron_results.query('Label == Prediction').shape[0]

print("Perceptron Results:")
perceptron_report = metrics.classification_report(perceptron_results['Label'], perc
print(metrics.classification_report(perceptron_results['Label'], perceptron_results

# perceptron_results.to_csv('PerceptronResults.csv', index=False)
```

```
Perceptron Results:
              precision    recall  f1-score   support

           1       0.76      0.54      0.63      4000
           2       0.61      0.51      0.55      3999
           3       0.61      0.89      0.72      3995

    accuracy                           0.65     11994
   macro avg       0.66      0.65      0.64     11994
weighted avg       0.66      0.65      0.64     11994
```

SVM

In [269…

```
from sklearn import svm

# machine = svm.SVC(kernel='linear', C=1)
machine = svm.LinearSVC()
machine.fit(train_population.iloc[0:train_population.shape[0], list(range(2,train_p
m_prediction = machine.predict(test_population.iloc[0:test_population.shape[0], lis
machine_results = pd.DataFrame(zip(test_population.iloc[0:test_population.shape[0],

svm_report = metrics.classification_report(machine_results['Label'], machine_result
print(metrics.classification_report(machine_results['Label'], machine_results['Pred
```

```
              precision    recall  f1-score   support

           1       0.72      0.72      0.72      4000
           2       0.64      0.61      0.63      3999
           3       0.78      0.81      0.79      3995

    accuracy                           0.71     11994
   macro avg       0.71      0.71      0.71     11994
weighted avg       0.71      0.71      0.71     11994
```

Logistic Regression

In [270…

```
from sklearn.linear_model import LogisticRegression

# regression = LogisticRegression(solver='lbfgs', multi_class='auto')
regression = LogisticRegression(max_iter=500)
regression.fit(train_population.iloc[0:train_population.shape[0], list(range(2,trai
r_prediction = regression.predict(test_population.iloc[0:test_population.shape[0],
regression_results = pd.DataFrame(zip(test_population.iloc[0:test_population.shape[
lr_report = metrics.classification_report(regression_results['Label'], regression_r
print(metrics.classification_report(regression_results['Label'], regression_results
```

```
              precision    recall  f1-score   support

           1       0.72      0.72      0.72      4000
           2       0.64      0.64      0.64      3999
           3       0.79      0.80      0.80      3995

    accuracy                           0.72     11994
   macro avg       0.72      0.72      0.72     11994
weighted avg       0.72      0.72      0.72     11994
```

Naive Bayes

In [271…

```
from sklearn.naive_bayes import MultinomialNB

bayes = MultinomialNB()
bayes.fit(train_population.iloc[0:train_population.shape[0], list(range(2,train_pop
b_prediction = bayes.predict(test_population.iloc[0:test_population.shape[0], list(
bayes_results = pd.DataFrame(zip(test_population.iloc[0:test_population.shape[0], 1
nb_report = metrics.classification_report(bayes_results['Label'], bayes_results['Pr
print(metrics.classification_report(bayes_results['Label'], bayes_results['Predicti
```

```
              precision    recall  f1-score   support

           1       0.70      0.68      0.69      4000
           2       0.60      0.65      0.63      3999
           3       0.78      0.75      0.76      3995

    accuracy                           0.69     11994
   macro avg       0.70      0.69      0.69     11994
weighted avg       0.70      0.69      0.69     11994
```

Summary

In [272…

```
print(f"Data Cleaning avg length (before, after): {before_cleaning}, {after_cleanin
print(f"Data Pre-processing avg length (before, after): {before_preproc}, {after_pr
print()
print("Model: (Precision, Recall, F1-Score)")
print(f"Perceptron Class 1: {perceptron_report['1']['precision']}, {perceptron_repo
print(f"Perceptron Class 2: {perceptron_report['2']['precision']}, {perceptron_repo
print(f"Perceptron Class 3: {perceptron_report['3']['precision']}, {perceptron_repo
print(f"Perceptron Average: {perceptron_report['weighted avg']['precision']}, {perc
print()
print(f"SVM Class 1: {svm_report['1']['precision']}, {svm_report['1']['recall']}, {
print(f"SVM Class 2: {svm_report['2']['precision']}, {svm_report['2']['recall']}, {
print(f"SVM Class 3: {svm_report['3']['precision']}, {svm_report['3']['recall']}, {
print(f"SVM Average: {svm_report['weighted avg']['precision']}, {svm_report['weight
print()
print(f"Regression Class 1: {lr_report['1']['precision']}, {lr_report['1']['recall'
print(f"Regression Class 2: {lr_report['2']['precision']}, {lr_report['2']['recall'
print(f"Regression Class 3: {lr_report['3']['precision']}, {lr_report['3']['recall'
print(f"Regression Average: {lr_report['weighted avg']['precision']}, {lr_report['w
print()
print(f"Naive Bayes Class 1: {nb_report['1']['precision']}, {nb_report['1']['recall
print(f"Naive Bayes Class 2: {nb_report['2']['precision']}, {nb_report['2']['recall
print(f"Naive Bayes Class 3: {nb_report['3']['precision']}, {nb_report['3']['recall
print(f"Naive Bayes Average: {nb_report['weighted avg']['precision']}, {nb_report['
```

```
Data Cleaning avg length (before, after): 293.66876666666667, 277.9042
Data Pre-processing avg length (before, after): 277.9042, 266.5327

Model: (Precision, Recall, F1-Score)
Perceptron Class 1: 0.764664059722716, 0.53775, 0.631439894319683
Perceptron Class 2: 0.6100420926037282, 0.5073768442110528, 0.5539931740614334
Perceptron Class 3: 0.6078565328778821, 0.8908635794743429, 0.7226395939086294
Perceptron Average: 0.6608805582841681, 0.6452392863098215, 0.6359949523107703

SVM Class 1: 0.7166872682323857, 0.72475, 0.7206960845245495
SVM Class 2: 0.641642688987706, 0.6134033508377095, 0.6272053183329073
SVM Class 3: 0.7804168686379059, 0.8060075093867334, 0.7930057874645979
SVM Average: 0.7128934114056873, 0.7146906786726697, 0.7136098488438022

Regression Class 1: 0.7209418837675351, 0.7195, 0.7202202202202203
Regression Class 2: 0.6383191595797899, 0.6381595398849712, 0.6382393397524071
Regression Class 3: 0.794955044955045, 0.7967459324155194, 0.7958494811851482
Regression Average: 0.7180466282162018, 0.7181090545272636, 0.7180772617880126

Naive Bayes Class 1: 0.7029419422025515, 0.675, 0.6886876673893637
Naive Bayes Class 2: 0.601802634619829, 0.6511627906976745, 0.6255104491952919
Naive Bayes Class 3: 0.781495033978045, 0.7484355444305382, 0.7646081063802583
Naive Bayes Average: 0.6953851230112715, 0.6915124228781058, 0.6929111506485374
```