# CSCI544-Assignment 1

-- by Ruijie Rao

# Report

## 1 Data Preparation

Using pandas library and its apply method to fulfill the requirement of labeling the ratings into 3 classes. Then, using the sample function in dataframes to make a balanced dataset with 20000 samples for each class.

## 2 Data Cleaning

- Average character length of reviews before data cleaning: 270.5612
- Average character length of reviews after data cleaning: 270.6393

Explanation: The character length average seem not to be changing much, but actually there has been a lot of work done. Regarding the removal of contractions, I have created my own dictionary of contractions, which actually increases the average character length. I have removed URLs and non-alhabetical characters using re.sub, but I let the question marks, quotation marks and exclamation marks remain. Since TFIDF cannot include these punctuations, I replaced them with phrases like "exclamationmark". This change actually increased the overall performance. As a result, the character length average of the reviews has not change much.

- Average character length of reviews before preprocessing: 270.6393
- Average character length of reviews after data cleaning: 250.57881666666665

Explanation: I found out that lemmatization works much accurately with a position tag provided by nltk. Moreover, I tried stemming but it did not have much influence. What is strange is that the models perform much better without the removal of stopwords. This is not only due to the removal of the word "not", which I have tried already. As a result, the removal process of stopwords has been recorded but not used.

## 3 Feature Extraction

Using the TfidfVectorizer from sklearn with a min-df=0.0008 ensures that the dictionary size is not unacceptable(can be more than 10k) and some miss-spelled words are removed. Although increasing the size does make better model fits, but it takes a lot more space and creates a lot more sparsity.

# 4 Perceptron

- <Class 1> Precision: 0.6667505030181087, Recall: 0.6632474355766825, F-1: 0.6649943559513357
- <Class 2> Precision: 0.5727951469902006, Recall: 0.6082755203171457, F-1: 0.5900024032684451
- <Class 3> Precision: 0.7672552166934189, Recall: 0.7229644567683388, F-1: 0.74445165476963
- [Overall Mean] Precision: 0.6689336222339094, Recall: 0.6648291375540557, F-1: 0.6664828046631369

Perceptron is the worst performing model of all. I have tried many parameters including the regularization penalties, but the results are not promising. I then put my hope on random_state, and makes a random-forest-like approach on the model by giving multiple tries to find the best random seed.

# 5 SVM

- <Class 1> Precision: 0.7050412021328163, Recall: 0.7277958468851639, F-1: 0.7162378431613936
- <Class 2> Precision: 0.6543472480723211, Recall: 0.6097621407333994, F-1: 0.6312684365781711
- <Class 3> Precision: 0.7734014101628981, Recall: 0.8018653894630703, F-1: 0.7873762376237623
- [Overall Mean] Precision: 0.7109299534560117, Recall: 0.7131411256938778, F-1: 0.7116275057877757

LinearSVC performs best out of all svm models in sklearn with a very good speed. Since the N_sample > N_features, dual is set to False. I adjusted C to optimize the test performance.

# 6 Logistic Regression

- <Class 1> Precision: 0.7200100175306787, Recall: 0.7192894671003253, F-1: 0.7196495619524406
- <Class 2> Precision: 0.6422155688622755, Recall: 0.6377601585728444, F-1: 0.6399801093983093
- <Class 3> Precision: 0.7884471117779445, Recall: 0.7948071590622636, F-1: 0.7916143610343961
- [Overall Mean] Precision: 0.7168908993902995, Recall: 0.7172855949118112, F-1: 0.717081344128382

Same as SVC, dual is set to false and C is adjusted. Availabe solver options for multiclass labeling all perform similarly.

## 7 Naive Bayes

- <Class 1> Precision: 0.6998950682056663, Recall: 0.6675006254691018, F-1: 0.6833141247278781
- <Class 2> Precision: 0.6, Recall: 0.6407333994053518, F-1: 0.6196980589503953
- <Class 3> Precision: 0.7679216090768437, Recall: 0.7506932190572221, F-1: 0.7592096876991715
- [Overall Mean] Precision: 0.6892722257608366, Recall: 0.6863090813105587, F-1: 0.6874072904591483

Tried all the models in skelarn.naive_bayes and found MultinomialNB the best choice. No parameters to adjust.

# Notebook

In [ ]:
```
import pandas as pd
import numpy as np
import nltk
import re
import csv
import re
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

In [ ]:
```
DATA_DIR = 'data/'
```

## Read Data

In [ ]:
```
raw_df = pd.read_table(DATA_DIR+'amazon_reviews_us_Beauty_v1_00.tsv.gz', compres
```

In [ ]:
```
raw_df.head()
```

| | marketplace | customer_id | review_id | product_id | product_parent | product_title |
|---|---|---|---|---|---|---|
| **0** | US | 1797882 | R3I2DHQBR577SS | B001ANOOOE | 2102612 | The Naked Bee Vitmin C Moisturizing Sunscreen ... |
| **1** | US | 18381298 | R1QNE9NQFJC2Y4 | B0016J22EQ | 106393691 | Alba Botanica Sunless Tanning Lotion, 4 Ounce |
| **2** | US | 19242472 | R3LIDG2Q4LJBAO | B00HU6UQAG | 375449471 | Elysee Infusion Skin Therapy Elixir, 2oz. |
| **3** | US | 19551372 | R3KSZHPAEVPEAL | B002HWS7RM | 255651889 | Diane D722 Color, Perm And Conditioner Process... |
| **4** | US | 14802407 | RAI2OIG50KZ43 | B00SM99KWU | 116158747 | Biore UV Aqua Rich Watery Essence SPF50+/PA+++... |

# Keep Reviews and Ratings

In [ ]:
```
df = raw_df[["star_rating","review_body"]]
```

In [ ]:
```
df = df.dropna()
```

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

In [ ]:
```
def label_class(x):
    if x<3:
        return 1
    if x>3:
        return 3
    else:
        return 2
```

In [ ]:
```
df["label"] = df["star_rating"].apply(label_class)
```

In [ ]:
```
df.head()
```

| | star_rating | review_body | label |
|---|---|---|---|
| **0** | 5 | Love this, excellent sun block!! | 3 |
| **1** | 5 | The great thing about this cream is that it do... | 3 |
| **2** | 5 | Great Product, I'm 65 years old and this is al... | 3 |
| **3** | 5 | I use them as shower caps & conditioning caps.... | 3 |
| **4** | 5 | This is my go-to daily sunblock. It leaves no ... | 3 |

In [ ]:
```
sampled_df = pd.concat([df[df["label"] == k].sample(n=20000) for k in range(1,4)
```

In [ ]:
```
len(sampled_df)
```

```
60000
```

# Data Cleaning

In [ ]:
```
contractions = {
"dont":"do not",
"ain't": "are not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
#...More deleted, find in .py
}
```

In [ ]:
```
# Print AVG length of reviews BEFORE data cleaning
print(f'AVG length of reviews BEFORE data cleaning: {sampled_df["review_body"].a
```

```
AVG length of reviews BEFORE data cleaning: 267.983
```

In [ ]:
```
def deconstract(x):
    tokens = x.split(' ')
    for i,token in enumerate(tokens):
        if token in contractions.keys():
            tokens[i] = contractions[token]
    return ' '.join(tokens)
```

In [ ]:
```
def data_cleaning(x):
    x = x.lower() #convert all reviews into lowercase
    x = re.sub(r'\s*https?://\S+(\s+|$)', '', x) #remove the HTML and URLs from
    x = re.sub(r'[^a-zA-Z ?!"]+', '', x) #remove non-alphabetical characters
    x = x.replace("!"," exclamationmark ")
    x = x.replace("?"," questionmark ")
    x = x.replace('"', ' quotationmark ')
    x = ' '.join(re.sub(r'\s', ' ', x).split()) #remove extra spaces
    x = deconstract(x)
    return x
```

In [ ]:
```
sampled_df["review_cleaned"] = sampled_df["review_body"].apply(data_cleaning)
```

In [ ]:
```
# Print AVG length of reviews AFTER cleaning
print(f'AVG length of reviews AFTER cleaning: {sampled_df["review_cleaned"].appl
```

```
AVG length of reviews AFTER cleaning: 268.2423333333333
```

# Pre-processing

## remove the stop words

In [ ]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
#stop_words.remove('not')
def remove_stopwords(x):
    return ' '.join([word for word in x.split(" ") if word not in stop_words])
```

In [ ]:
```
def only_stopwords(x):
    return ' '.join([word for word in x.split(" ") if word in stop_words])
sampled_df["stopwords"] = sampled_df["review_cleaned"].apply(only_stopwords)
```

## perform lemmatization

In [ ]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def removeStopwords_lemmatization(x):
    return ' '.join([lemmatizer.lemmatize(word, pos=nltk.corpus.wordnet.synsets(
```

In [ ]:
```
wd = nltk.corpus.wordnet
def ultimate_preprocess(x):
    tokens = []
    for word in x.split(" "):
        if word not in stop_words:
            try:
                pos = wd.synsets(word)[0].pos()
                tokens.append(lemmatizer.lemmatize(word, pos=pos))
            except IndexError:
                tokens.append(lemmatizer.lemmatize(word))
    return ' '.join(tokens)
```

In [ ]:
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
def ultimate_preprocess_withstopwords(x):
    tokens = []
    for word in x.split(" "):
        try:
            pos = wd.synsets(word)[0].pos()
            tokens.append(ps.stem(lemmatizer.lemmatize(word, pos=pos)))
        except IndexError:
            tokens.append(ps.stem(lemmatizer.lemmatize(word)))
    return ' '.join(tokens)
```

In [ ]:
```
sampled_df["review_processed"] = sampled_df["review_cleaned"].apply(ultimate_pre
```

In [ ]:
```
# Print AVG length of reviews AFTER preprocessed
print(f'AVG length of reviews AFTER preprocessed: {sampled_df["review_processed"
```

```
AVG length of reviews AFTER preprocessed: 248.33835
```

In [ ]:
```
sampled_df.head()
```

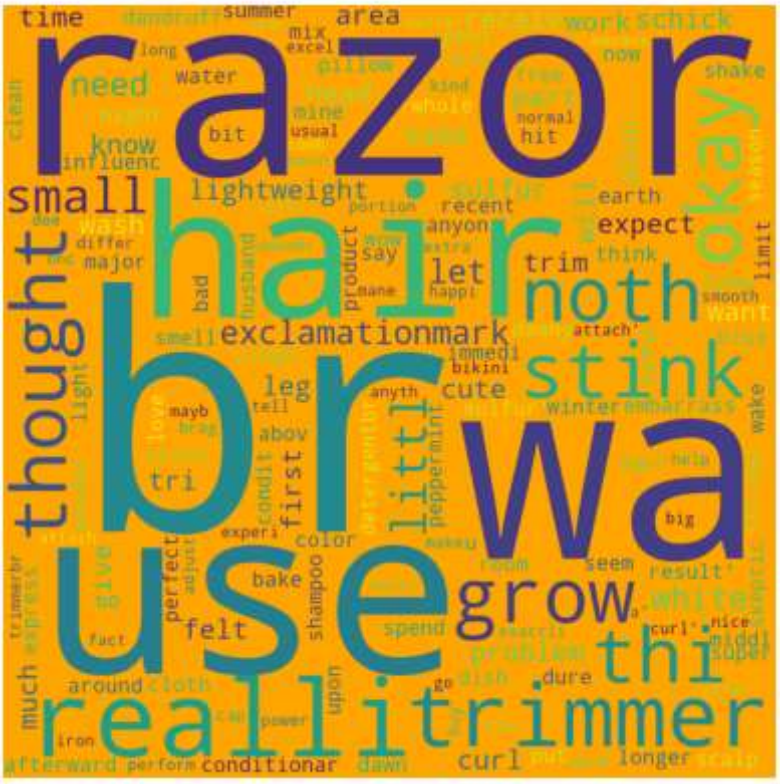
| | star_rating | review_body | label | review_cleaned | stopwords | review_processed |
|---|---|---|---|---|---|---|
| **1238967** | 1 | I don't know what I was expecting, because the... | 1 | i do not know what i was expecting because the... | i do not what i was because these are and as i... | i do not know what i wa expect becaus these th... |
| **709826** | 1 | A bit disappointed. The brush is so small. So ... | 1 | a bit disappointed the brush is so small so ha... | a the is so so to while you up your with your ... | a bit disappoint the brush be so small so hard... |
| **4840765** | 1 | This made my mouth really sore after about a w... | 1 | this made my mouth really sore after about a w... | this my after about a or so of it i my of i it... | thi make my mouth realli sore after about a we... |
| **88958** | 1 | Not worth the purchase unfortunately. No results | 1 | not worth the purchase unfortunately no results | not the no | not worth the purchas unfortun no result |
| **1846121** | 1 | I was given this product to try for free. Firs... | 1 | i was given this product to try for free first... | i was this to for the we are to for me when i ... | i wa given thi product to tri for free first t... |

# Exploration

In [ ]:
```
def word_exploration(text_series):
    plt.rcParams['figure.figsize'] = (5,5)
    plt.style.use('fast')
    wc = WordCloud(background_color='orange', width=800, height=800).generate(st
    plt.imshow(wc)
    plt.axis('off')
    plt.show()
```

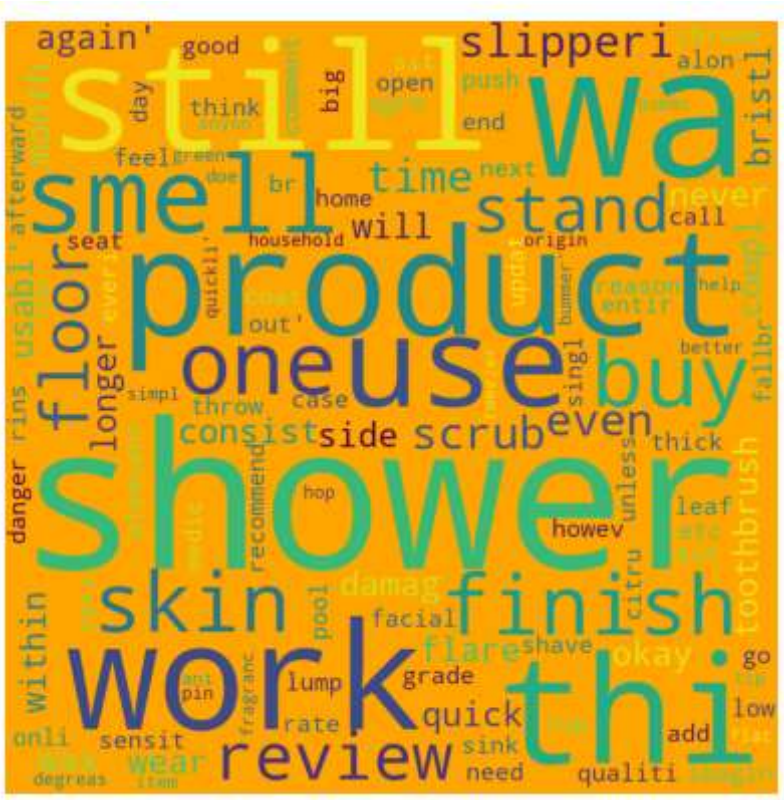
In [ ]:
```
neutral = sampled_df[sampled_df["label"] == 2]["review_processed"].values
poitive = sampled_df[sampled_df["label"] == 3]["review_processed"].values
negative = sampled_df[sampled_df["label"] == 1]["review_processed"].values
```

In [ ]:
```
word_exploration(neutral)
```




In [ ]:
```
word_exploration(poitive)
```


In [ ]:
```
word_exploration(negative)
```




# TF-IDF Feature Extraction

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = sampled_df["review_processed"].values
vectorizer = TfidfVectorizer(min_df=0.0008)
feature = vectorizer.fit_transform(corpus)
```

In [ ]:
```
feature_df = pd.DataFrame(feature.toarray(), columns=vectorizer.get_feature_name
```

In [ ]:
```
vectorizer.get_feature_names_out()
```

```
array(['abil', 'abl', 'about', ..., 'zero', 'zinc', 'zipper'],
      dtype=object)
```

In [ ]:
```
len(vectorizer.get_feature_names_out())
```

```
2289
```

In [ ]:
```
feature_df.head()
```

| | about | above | after | again | against | all | am | an | and | any |
|---|---|---|---|---|---|---|---|---|---|---|
| **3771162** | 0.000000 | 0.0 | 0.00000 | 0.0 | 0.0 | 0.00000 | 0.000000 | 0.000000 | 0.213711 | 0.0 |
| **5091345** | 0.161406 | 0.0 | 0.00000 | 0.0 | 0.0 | 0.13823 | 0.000000 | 0.167548 | 0.221709 | 0.0 |
| **1818307** | 0.350219 | 0.0 | 0.32249 | 0.0 | 0.0 | 0.00000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **2825695** | 0.000000 | 0.0 | 0.00000 | 0.0 | 0.0 | 0.00000 | 0.084873 | 0.104643 | 0.230783 | 0.0 |
| **1469692** | 0.000000 | 0.0 | 0.00000 | 0.0 | 0.0 | 0.00000 | 0.000000 | 0.000000 | 0.390163 | 0.0 |

5 rows × 130 columns

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(feature_df, sampled_df[['lab
```

In [ ]:
```
import pickle
with open(DATA_DIR+"stemmed/data.pkl","wb") as file:
    pickle.dump([X_train, X_test, y_train, y_test], file)
```

# Perceptron

In [ ]:
```
from sklearn.metrics import f1_score, precision_score, recall_score
def evaluation(y_pred, y_true):
    prc = precision_score(y_true, y_pred, average=None)
    recall = recall_score(y_true, y_pred, average=None)
    f1 = f1_score(y_true, y_pred, average=None)
    for cls in range(1,4):
        print(f'<Class {cls}> Precision: {prc[cls-1]}, Recall: {recall[cls-1]},
    print(f'<Overall Mean> Precision: {np.mean(prc)}, Recall: {np.mean(recall)},
```

In [ ]:
```
from sklearn.linear_model import Perceptron
```

In [ ]:
```
def find_best_randseed(N=10):
    best_f1 = 0
    for i in range(N):
        seed = np.random.randint(0,1000)
        perceptron_md = Perceptron(tol=1e-3, random_state=seed, penalty='elastic
        perceptron_md.fit(X_train, y_train.values.ravel())
        f1 = np.mean(f1_score(y_test, perceptron_md.predict(X_test), average=Nor
        if f1 > best_f1:
            best_f1 = f1
            best_seed = seed
    return best_seed
```

In [ ]:
```
best_seed = find_best_randseed(20)
perceptron_md = Perceptron(tol=1e-3, random_state=best_seed)
perceptron_md.fit(X_train, y_train.values.ravel())
evaluation(perceptron_md.predict(X_test), y_test)
```

```
<Class 1> Precision: 0.6667505030181087, Recall: 0.6632474355766825, F-1: 0.664
9943559513357
<Class 2> Precision: 0.5727951469902006, Recall: 0.6082755203171457, F-1: 0.590
0024032684451
<Class 3> Precision: 0.7672552166934189, Recall: 0.7229644567683388, F-1: 0.744
45165476963
<Overall Mean> Precision: 0.6689336222339094, Recall: 0.6648291375540557, F-1:
0.6664828046631369
```

# SVM

In [ ]:
```
from sklearn.svm import SVC, LinearSVC
```

In [ ]:
```
svm_md = LinearSVC(random_state=0, dual=False, C=0.05)
svm_md.fit(X_train, y_train.values.ravel())
evaluation(svm_md.predict(X_test), y_test)
```

```
<Class 1> Precision: 0.7050412021328163, Recall: 0.7277958468851639, F-1: 0.716
2378431613936
<Class 2> Precision: 0.6543472480723211, Recall: 0.6097621407333994, F-1: 0.631
2684365781711
<Class 3> Precision: 0.7734014101628981, Recall: 0.8018653894630703, F-1: 0.787
3762376237623
<Overall Mean> Precision: 0.7109299534560117, Recall: 0.7131411256938778, F-1:
0.7116275057877757
```

# Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
LR_md = LogisticRegression(tol=1e-3, random_state=0, solver='saga', dual=False,
LR_md.fit(X_train, y_train.values.ravel())
evaluation(LR_md.predict(X_test), y_test)
```

```
<Class 1> Precision: 0.7200100175306787, Recall: 0.7192894671003253, F-1: 0.719
6495619524406
<Class 2> Precision: 0.6422155688622755, Recall: 0.6377601585728444, F-1: 0.639
9801093983093
<Class 3> Precision: 0.7884471117779445, Recall: 0.7948071590622636, F-1: 0.791
6143610343961
<Overall Mean> Precision: 0.7168908993902995, Recall: 0.7172855949118112, F-1:
0.717081344128382
```

# Naive Bayes

In [ ]:
```
from sklearn.naive_bayes import MultinomialNB, CategoricalNB, BernoulliNB, Compl
```

In [ ]:
```
NB_md = MultinomialNB()
NB_md.fit(X_train, y_train.values.ravel())
evaluation(NB_md.predict(X_test), y_test)
```

```
<Class 1> Precision: 0.6998950682056663, Recall: 0.6675006254691018, F-1: 0.683
3141247278781
<Class 2> Precision: 0.6, Recall: 0.6407333994053518, F-1: 0.6196980589503953
<Class 3> Precision: 0.7679216090768437, Recall: 0.7506932190572221, F-1: 0.759
2096876991715
<Overall Mean> Precision: 0.6892722257608366, Recall: 0.6863090813105587, F-1:
0.6874072904591483
```