HW1 Report - Menghan Xu
Python version: 3.8.8

Brief:

Loaded data from gz file directly and suppressed bad line warning. Before sampling 200,000 reviews, Idropped rows with empty reviews so they wouldn't get selected.

When sampling, I used random seed for dataset consistency. BeautifulSoup is easy to use with html parser along with regex substitution, I was able to remove the html tags and any URL from the review body.I was not able to install contraction or pycontractions using pip, so I had to create a json based on the listed contractions in English from Wikipedia as a dictionary for contraction, then try to replace any match in each review. lowercase is straightforward with str.lower(), and I used regex again for remove non alphabetical and extra whitespaces. The steps in cleaning were done in such an order that the contractions don't get altered, such as "i'm" becomes "im".

For pre-processing, I used nltk stop word library to remove any stopwords and join the rest of the words from review with a single whitespace. For lemmatization, I first tried to look up the POS tag for each word, then lemmatize it with the tags, but it took about 10 mins each time for the 200,000 reviews, and after comparing the model results and feature counts, I turned to just passing the word to lemmatize(), which is much faster and results are nearly the same.

I did not split the training and testing dataset until the pre-processing was done, and I only used 160,000 train data with TfidfVector, because in real world problems, we cannot train with the data(test data in this case) we haven't collected. The total features turned out to be just above 50,000, also only single gram.

The model training steps are similar by fitting the vector matrix and classification to the model and predict on both train and test data set. I used learning rate 0.1 in perceptron and 1000 iterations, the result was fast so I did not experiment more on it. The SVM model fit was extremely slow with default options, so I turned to LinearSvm with tolerance 1e-5 for faster completion. In logistic regression, I just used 500 iterations. For MNB, I just kept it default.

Outputs:

Statistics of three classes, (positive, star=3, negative) :
3856492,349547,668848
Raw data classification:
Positive review: 3856492
Negative review: 668848

Average length of reviews before and after data cleaning:
324.359335,310.915845

Average length of reviews before and after data preprocessing:
310.915845, 191.15878

Three sample reviews in raw format:

1. Very Nice!! Exactly as described!!!!!
2. Easy to use. Ice cream in 20 minutes, couldn't be better.
3. Even though this product works great, the edge can break or crack easily. The edge is where the glass doubles back to create the space between the two walls. When you have any liquids in the cup, all that weight is being cantilevered from the outer wall to the inner wall, i.e. suspended. So, if you happen to bump or shake the cup while it is full, that puts a lot of stress on that edge. I've had 1 cup chip. I also bought the water jug but that one just broke when I bumped it against the sink after filling it. Yea, the worst possible scenario. So, if you are willing to pay for the great looks and functionality buy this but do expect that they won't last as other cups that cost less.

The double wall really does insulate the contents, hot or cold. So, your contents last longer and your hand does not get hot or cold. Great for ice cream and tea. Their tea pot is also very good. I haven't broken that one.....yet.

Three sample reviews after cleaning & preprocessing:
1. nice exactly described
2. easy use ice cream minute could better
3. even though product work great edge break crack easily edge glass double back create space two wall liquid cup weight cantilevered outer wall inner wall e suspended happen bump shake cup full put lot stress edge cup chip also bought water jug one broke bumped sink filling yea worst possible scenario willing pay great look functionality buy expect last cup cost le double wall really insulate content hot cold content last longer hand get hot cold great ice cream tea tea pot also good broken one yet

Model report:

Accuracy, Precision, Recall, and f1-score for training and testing split for Perceptron:
0.89514375,0.8751737658171449,0.9215438508695108,0.8977604436454494,0.8549,0.8358918817103033,0.8844544095665172,0.8594877257541278

Accuracy, Precision, Recall, and f1-score for training and testing split for SVM:
0.9324,0.9334688040942274,0.9310396597022395,0.93225264951269,0.8939,0.894899690587883,0.8934728450423518,0.894185698613743

Accuracy, Precision, Recall, and f1-score for training and testing split for Logistic Regression:
0.91286875,0.9161820910960027,0.9087201301138497,0.9124358547569547,0.896475,0.8996437352601736,0.8933233682112606,0.8964724118102952

Accuracy, Precision, Recall, and f1-score for training and testing split for Naive Bayes:
0.88603125,0.8907566218664099,0.8797572876266734,0.8852227880130671,0.8674,0.8739363857374393,0.8597409068261086,0.8667805294619984

HW1-CSCI544

September 8, 2021

[49]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
import string
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split
from sklearn import metrics

import warnings
warnings.filterwarnings("ignore", category=UserWarning, module='bs4')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\xmh91\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[ ]:
```
#! pip install bs4 # in case you don't have it installed

# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/
 ↪amazon_reviews_us_Kitchen_v1_00.tsv.gz
```

0.1 Read Data

[50]:
```
raw_df = pd.read_csv("amazon_reviews_us_Kitchen_v1_00.tsv.gz", sep='\t',␣
 ↪error_bad_lines=False, warn_bad_lines=False)
```

0.2 Data report

[51]:
```
pos_review_count = len(raw_df[ raw_df['star_rating'] > 3 ])
mid_review_count= len(raw_df[ raw_df['star_rating'] == 3 ])
neg_review_count = len(raw_df[ raw_df['star_rating'] < 3 ])

print(f"""
Statistics of three classes, (positive, star=3, negative) :
{pos_review_count},{mid_review_count},{neg_review_count}
""")
```

```
#pd.value_counts(df['star_rating']).plot.bar()
```

Statistics of three classes, (positive, star=3, negative) :
3856492,349547,668848

0.3 Keep Reviews and Ratings

[52]:
```
df = raw_df[['star_rating', 'review_body']].dropna(subset=['review_body'])
```

1 Labelling Reviews:

1.1 The reviews with rating 4,5 are labelled to be 1 and 1,2 are labelled as 0. Discard the reviews with rating 3'

[53]:
```
# Drop rating 3 rows
three_indexes = df[ df['star_rating'] == 3 ].index
df.drop(three_indexes, inplace = True)
# Lebal samples according to rating
df.loc[df['star_rating'] <= 2, 'star_rating'] = 0
df.loc[df['star_rating'] >= 4, 'star_rating'] = 1
df.columns = ['label', 'review']
```

We select 200000 reviews randomly with 100,000 positive and 100,000 negative reviews.

[54]:
```
# select reviews
df_pos = df.loc[df['label'] == 1].sample(100000, random_state=1)
df_neg = df.loc[df['label'] == 0].sample(100000, random_state=2)

# concatenate both positive and negative review df together for cleaning and␣
 ↪preprocessing
df_data = df_pos.append(df_neg)
```

[55]:
```
# sample 3 reviews
print("Report samples without cleaning & preprocessing:")
df_3_sample = df_data.sample(3, random_state=6)
sample_counter=1
for rev in df_3_sample["review"]:
    print(f"\t{sample_counter}. {rev}")
    sample_counter += 1
```

```
Report samples without cleaning & preprocessing:
        1. Very Nice!!  Exactly as described!!!!!
        2. Easy to use.  Ice cream in 20 minutes, couldn't be better.
        3. Even though this product works great, the edge can break or crack
easily. The edge is where the glass doubles back to create the space between the
```

two walls. When you have any liquids in the cup, all that weight is being cantilevered from the outer wall to the inner wall, i.e. suspended. So, if you happen to bump or shake the cup while it is full, that puts a lot of stress on that edge. I've had 1 cup chip. I also bought the water jug but that one just broke when I bumped it against the sink after filling it. Yea, the worst possible scenario. So, if you are willing to pay for the great looks and functionality buy this but do expect that they won't last as other cups that cost less.

The double wall really does insulate the contents, hot or cold. So, your contents last longer and your hand does not get hot or cold. Great for ice cream and tea. Their tea pot is also very good. I haven't broken that one…yet.

2 Data Cleaning

2.1 Convert the all reviews into the lower case.

[56]:
```
# Average review length prior to data cleaning

len_before_clean = df_data["review"].apply(len).mean()
```

[57]:
```
df_data['review'] = df_data['review'].str.lower()
```

[58]:
```
# 3 samples
df_3_sample['review'] = df_3_sample['review'].str.lower()
```

2.2 remove the HTML and URLs from the reviews

[59]:
```
def remove_html_url(s):
    # parse html
    soup = BeautifulSoup(s, "html.parser")

    for data in soup(['style', 'script']):
        # remove tags
        data.decompose()
    # replace url with empty string and return
    return re.sub(r"http\S+", "", ' '.join(soup.stripped_strings))

# Remove HTML markups and URL in text format
df_data['review'] = [ remove_html_url(review) for review in df_data['review'] ]
```

[60]:
```
# 3 samples
df_3_sample['review'] = [ remove_html_url(review) for review in␣
 ↪df_3_sample['review'] ]
```

2.3 Contractions

2.3.1 **unable to isntall pycontraction, nor find any workaround library

[61]:
```
contractions = {
        "a'ight": "alright",
        "ain't": "am not",
        "amn't": "am not",
        "arencha": "are not you",
        "aren't": "are not",
        "‘bout": "about",
        "cannot": "can not",
        "can't": "cannot",
        "cap’n": "captain",
        "cause": "because",
        "’cept": "except",
        "could've": "could have",
        "couldn't": "could not",
        "couldn't've": "could not have",
        "dammit": "damn it",
        "daren't": "dare not",
        "daresn't": "dare not",
        "dasn't": "dare not",
        "didn't": "did not",
        "doesn't": "does not",
        "don't": "do not",
        "dunno": "do not know",
        "d'ye": "did you",
        "e'en": "even",
        "e'er": "ever",
        "em": "them",
        "everybody's": "everybody is",
        "everyone's": "everyone is",
        "fo’c’sle": "forecastle",
        "’gainst": "against",
        "g'day": "good day",
        "gimme": "give me",
        "giv'n": "given",
        "gonna": "going to",
        "gon't": "go not",
        "gotta": "got to",
        "hadn't": "had not",
        "had've": "had have",
        "hasn't": "has not",
        "haven't": "have not",
        "he'd": "he would",
        "he'll": "he will",
        "helluva": "hell of a",
```

```
        "he's": "he is",
        "here's": "here is",
        "how'd": "how did",
        "howdy": "how do you do",
        "how'll": "how will",
        "how're": "how are",
        "how's": "how is",
        "i'd": "i would",
        "i'd've": "i would have",
        "i'll": "i will",
        "i'm": "i am",
        "imma": "i am about to",
        "i'm'o": "i am going to",
        "innit": "is it not",
        "ion": "i do not",
        "i've": "i have",
        "isn't": "is not",
        "it'd": "it would",
        "it'll": "it will",
        "it's": "it is",
        "iunno": "i do not know",
        "kinda": "kind of",
        "let's": "let us",
        "ma'am": "madam",
        "mayn't": "may not",
        "may've": "may have",
        "methinks": "i think",
        "mightn't": "might not",
        "might've": "might have",
        "mustn't": "must not",
        "mustn't've": "must not have",
        "must've": "must have",
        "'neath": "beneath",
        "needn't": "need not",
        "nal": "and all",
        "ne'er": "never",
        "o'clock": "of the clock",
        "o'er": "over",
        "ol'": "old",
        "oughtn't": "ought not",
        "'round": "around",
        "s": "is",
        "shalln't": "shall not",
        "shan't": "shall not",
        "she'd": "she would",
        "she'll": "she will",
        "she's": "she is",
```

```
        "should've": "should have",
        "shouldn't": "should not",
        "shouldn't've": "should not have",
        "somebody's": "somebody is",
        "someone's": "someone is",
        "something's": "something is",
        "so're": "so are",
        "so’s": "so has",
        "so’ve": "so have",
        "that'll": "that will",
        "that're": "that are",
        "that's": "that is",
        "that'd": "that would",
        "there'd": "there would",
        "there'll": "there will",
        "there're": "there are",
        "there's": "there is",
        "these're": "these are",
        "these've": "these have",
        "they'd": "they would",
        "they'll": "they will",
        "they're": "they are",
        "they've": "they have",
        "this's": "this is",
        "those're": "those are",
        "those've": "those have",
        "thout": "without",
        "’til": "until",
        "tis": "it is",
        "to've": "to have",
        "twas": "it was",
        "tween": "between",
        "twere": "it were",
        "wanna": "want to",
        "wasn't": "was not",
        "we'd": "we would",
        "we'd've": "we would have",
        "we'll": "we will",
        "we're": "we are",
        "we've": "we have",
        "weren't": "were not",
        "whatcha": "what are you",
        "what'd": "what did",
        "what'll": "what will",
        "what're": "what are/what were",
        "what's": "what is",
        "what've": "what have",
```

```
        "when's": "when is",
        "where'd": "where did",
        "where'll": "where will",
        "where're": "where are",
        "where's": "where is",
        "where've": "where have",
        "which'd": "which had",
        "which'll": "which will",
        "which're": "which are",
        "which's": "which is",
        "which've": "which have",
        "who'd": "who would",
        "who'd've": "who would have",
        "who'll": "who will",
        "who're": "who are",
        "who's": "who is",
        "who've": "who have",
        "why'd": "why did",
        "why're": "why are",
        "why's": "why is",
        "willn't": "will not",
        "won't": "will not",
        "wonnot": "will not",
        "would've": "would have",
        "wouldn't": "would not",
        "wouldn't've": "would not have",
        "y'all": "you all",
        "y'all'd've": "you all would have",
        "y'all'd'n't've": "you all would not have",
        "y'all're": "you all are",
        "y'all'ren't": "you all are not",
        "y'at": "you at",
        "yes'm": "yes madam",
        "yessir": "yes sir",
        "you'd": "you would",
        "you'll": "you will",
        "you're": "you are",
        "you've": "you have",
        "yrs": "years",
        "ur" : "your",
        "urs" : "yours"
}

def contractionfunction(s):
    for word in s.split(" "):
        if word in contractions:
            s = s.replace(word, contractions[word])
```

```
    return s

df_data['review'] = [ contractionfunction(review) for review in␣
 ↪df_data['review'] ]
```

[62]:
```
# 3 samples
df_3_sample['review'] = [ contractionfunction(review) for review in␣
 ↪df_3_sample['review'] ]
```

2.4 Remove non-alphabetical characters

[63]:
```
def remove_non_alphabetical(s):
    # remove single quote in word like " husband's "
    s = re.sub(r'\'', '', s)

    # replace non-alphabetical by whitespace
    s = re.sub(r"[^a-zA-Z]", ' ', s)

    # remove punctuation and return
    #return ' '.join([word.strip(string.punctuation) for word in s.split(" ")])
    return s

df_data['review'] = [ remove_non_alphabetical(review) for review in␣
 ↪df_data['review'] ]
```

[64]:
```
# 3 samples
df_3_sample['review'] = [ remove_non_alphabetical(review) for review in␣
 ↪df_3_sample['review'] ]
```

2.5 Remove the extra spaces between the words.

[65]:
```
df_data['review'] = [ re.sub(r'\s+', ' ', review) for review in␣
 ↪df_data['review'] ]
```

[66]:
```
# 3 samples
df_3_sample['review'] = [ re.sub(r'\s+', ' ', review) for review in␣
 ↪df_3_sample['review'] ]
```

[67]:
```
# Average review length after data cleaning

len_after_clean = df_data["review"].apply(len).mean()
print(f"""Average length of reviews before and after data cleaning:
{len_before_clean},{len_after_clean}""")
```

```
Average length of reviews before and after data cleaning:
324.359335,310.915845
```

3 Pre-processing

3.1 remove the stop words

[68]:
```
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

def remove_stopwords(s):
    # only keep words not in stopwords set
    filtered_words = [word for word in s.split(" ") if word not in stop_words]
    return " ".join(filtered_words)

df_data['review'] = [ remove_stopwords(review) for review in df_data['review'] ]
```

[69]:
```
#3 samples
df_3_sample['review'] = [ remove_stopwords(review) for review in␣
 ↪df_3_sample['review'] ]
```

3.2 perform lemmatization

[70]:
```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()

# simple lemmatize
def easy_lemmatize(s):
    lemmatized_words = [lemmatizer.lemmatize(word) for word in s.split(" ")]
    return " ".join(lemmatized_words)

# use simple lemmatize
df_data['review'] = [ easy_lemmatize(review) for review in df_data['review'] ]
#print("--- %s seconds ---" % (time.time() - start_time))
```

[71]:
```
#3 samples
df_3_sample['review'] = [ easy_lemmatize(review) for review in␣
 ↪df_3_sample['review'] ]
```

[72]:
```
# Average review length after data cleaning & preprocessing

len_after_prep = df_data["review"].apply(len).mean()
print(f"""Average length of reviews before and after data preprocessing:
{len_after_clean}, {len_after_prep}""")
```

```
Average length of reviews before and after data preprocessing:
310.915845, 191.15878
```

[73]:
```
print("Three sample reviews after cleaning & preprocessing:")
sample_counter=1
for rev in df_3_sample["review"]:
    print(f"\t{sample_counter}. {rev}")
    sample_counter += 1
```

```
Three sample reviews after cleaning & preprocessing:
        1. nice exactly described
        2. easy use ice cream minute could better
        3. even though product work great edge break crack easily edge glass
double back create space two wall liquid cup weight cantilevered outer wall
inner wall e suspended happen bump shake cup full put lot stress edge cup chip
also bought water jug one broke bumped sink filling yea worst possible scenario
willing pay great look functionality buy expect last cup cost le double wall
really insulate content hot cold content last longer hand get hot cold great ice
cream tea tea pot also good broken one yet
```

4 TF-IDF Feature Extraction

[74]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Split the data to train and test dataset
df_x = df_data['review']
df_y = df_data['label']

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, random_state =␣
↪3, test_size = 0.2)

# get tf-idf vectors from training data
vectorizer = TfidfVectorizer()
x_train_vec = vectorizer.fit_transform(x_train)

y_train=y_train.astype('int')

# transform test dataset based on train dataset tf-idf vectors
x_test_vec = vectorizer.transform(x_test)

#x_train_vec.shape
```

5 Report function

[76]:
```
# report classification metrics based on targets and predictions
def report_metrics(targets, predictions):

    accuracy = metrics.accuracy_score(targets, predictions)
    precision = metrics.precision_score(targets, predictions)
```

```
    recall = metrics.recall_score(targets, predictions)
    f1_score = metrics.f1_score(targets, predictions)

    return f"{accuracy},{precision},{recall},{f1_score}"
```

6 Perceptron

[79]:
```
from sklearn.linear_model import Perceptron

# set iteration and learning rate
max_iter = 1000
eta0 = 0.1

# Train the model with train dataset and train targets
ppn_clf = Perceptron(max_iter=max_iter, eta0=eta0, random_state=1)
ppn_clf.fit(x_train_vec, y_train)

# predict on train dataset
y_pred = ppn_clf.predict(x_train_vec)

# Classification Report - Train
output = report_metrics(y_train, y_pred)+","

# predict on test dataset
y_pred = ppn_clf.predict(x_test_vec)

# Classification Report - Test
output += report_metrics(y_test, y_pred)

print("Accuracy, Precision, Recall, and f1-score for training and testing split␣
 ↪for Perceptron:")
print(output)
```

```
Accuracy, Precision, Recall, and f1-score for training and testing split for
Perceptron:
0.89514375,0.8751737658171449,0.9215438508695108,0.8977604436454494,0.8549,0.835
8918817103033,0.8844544095665172,0.8594877257541278
```

7 SVM

[80]:
```
from sklearn.svm import LinearSVC

# Train the model with train dataset and train targets
svm = LinearSVC(random_state=0, tol=1e-5)
svm.fit(x_train_vec, y_train)
```

```
# predict on train dataset
y_pred = svm.predict(x_train_vec)

# Classification Report - Train
output = report_metrics(y_train, y_pred)+","

# predict on test dataset
y_pred = svm.predict(x_test_vec)

# Classification Report - Test
output += report_metrics(y_test, y_pred)

print("Accuracy, Precision, Recall, and f1-score for training and testing split␣
 ↪for SVM:")
print(output)
```

```
Accuracy, Precision, Recall, and f1-score for training and testing split for
SVM:
0.9324,0.9334688040942274,0.9310396597022395,0.93225264951269,0.8939,0.894899690
587883,0.8934728450423518,0.894185698613743
```

8 Logistic Regression

[81]:
```
from sklearn.linear_model import LogisticRegression

# Train the model with train dataset and train targets
lr_clp = LogisticRegression(random_state=0, max_iter=500)
lr_clp.fit(x_train_vec, y_train)

# predict on train dataset
y_pred = lr_clp.predict(x_train_vec)

# Classification Report - Train
output = report_metrics(y_train, y_pred)+","

# predict on test dataset
y_pred = lr_clp.predict(x_test_vec)

# Classification Report - Test
output += report_metrics(y_test, y_pred)

print("Accuracy, Precision, Recall, and f1-score for training and testing split␣
 ↪for Logistic Regression:")
print(output)
```

```
Accuracy, Precision, Recall, and f1-score for training and testing split for
Logistic Regression:
```

0.91286875,0.9161820910960027,0.9087201301138497,0.9124358547569547,0.896475,0.8996437352601736,0.8933233682112606,0.8964724118102952

9 Naive Bayes

[82]:
```
from sklearn.naive_bayes import MultinomialNB

# Train the model with train dataset and train targets
naive_bayes_clf = MultinomialNB()
naive_bayes_clf.fit(x_train_vec, y_train)

# predict on train dataset
y_pred = naive_bayes_clf.predict(x_train_vec)

# Classification Report - Train
output = report_metrics(y_train, y_pred)+","

# predict on test dataset
y_pred = naive_bayes_clf.predict(x_test_vec)

# Classification Report - Test
output += report_metrics(y_test, y_pred)

print("Accuracy, Precision, Recall, and f1-score for training and testing split␣
 ↪for Naive Bayes:")
print(output)
```

```
Accuracy, Precision, Recall, and f1-score for training and testing split for
Naive Bayes:
0.88603125,0.8907566218664099,0.8797572876266734,0.8852227880130671,0.8674,0.873
9363857374393,0.8597409068261086,0.8667805294619984
```