CSCI544: Homework Assignment No .1

1. Dataset Preparation

We will use the Amazon reviews dataset which contains real reviews for kitchen products sold on Amazon. The dataset is downloadable at: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Kitchen_v1_00.tsv.gz

A: Downloaded the file from the link given above.

a) Read the data as a Pandas frame using Pandas package and only keep the Reviews and Ratings _elds in the input data frame to generate data. Our goal is to train sentiment analysis classifiers that can predict the sentiment (positive/negative) for a given review.

Used pandas package to read the tsv file. In this dataset some of the rows were corrupted or had bad rows, therefore, we needed to mention a parameter called “error_bad_lines” to FALSE which skip all the bad rows.

Read Data

```
df = pd.read_csv("../amazon_reviews_us_Kitchen_v1_00.tsv", sep = "\t", error_bad_lines=False, warn_bad_lines=False)
```

As dataset is too large and we only need reviews and ratings, we filtered out our dataframe keeping onl y ratings and reviews. There might be some cells which have missing values so to get rid of that we dr opped those rows so that our dataframe is correct and consistent.

Keep Reviews and Ratings

```
df = df[['star_rating', 'review_body']]
print("REVIEWS SAMPPLE WITH RATING")
print(df.head(3))
print("STATISIICS OF THE REVIEWS: ")
print("rating 1: ", len(df[df.star_rating == 1]))
print("rating 2: ", len(df[df.star_rating == 2]))
print("rating 3: ", len(df[df.star_rating == 3]))
print("rating 4: ", len(df[df.star_rating == 4]))
print("rating 5: ", len(df[df.star_rating == 5]))
df = df.dropna()
```

```
REVIEWS SAMPPLE WITH RATING
   star_rating                                        review_body
0          5.0                Beautiful.  Looks great on counter.
1          5.0  I personally have 5 days sets and have also bo...
2          5.0  Fabulous and worth every penny. Used for clean...
STATISIICS OF THE REVIEWS:
rating 1:  426900
rating 2:  241948
rating 3:  349547
rating 4:  731733
rating 5:  3124759
```

To have binary class labels we added an extra column in our dataframe called "labels" The mapping of labels has been defined as follows:

If any reviews rating is more than 3 it will be classified as 1.
If any reviews rating is less than 3 it will be classified as 0.
And rest of the reviews which have rating equals 3 will be discarded.

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

```
df['label'] = df.apply(lambda row: 1 if row.star_rating > 3 else 0, axis = 1)
r0 = len(df[df.label == 0])
r1 = len(df[df.label == 1])
r3 = len(df[df.star_rating == 3])
df = df[df.star_rating != 3]
print(r0, ", ", r1, ", ", r3)
```

```
1018348 ,  3856296 ,  349539
```

As the dataset is too large, we are going to take only 200000 rows which will have 50% positive sampl es and 50% negative samples. After having those samples, we are going to shuffle the dataset so that w hen we are splitting the model can be trained on both the samples.

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

```
positive = df[df['label'] == 1]
negative = df[df['label'] == 0]
positive = positive.sample(n = 100000)
negative = negative.sample(n = 100000)
dataset = pd.concat([positive, negative])
dataset = dataset.reset_index(drop = True)
train=dataset.sample(frac=0.8,random_state=200)
test = dataset.drop(train.index)
train = train.reset_index(drop = True)
test = test.reset_index(drop = True)
```

We are splitting our dataset into training and testing dataset with 80 -20 split.

2. Preprocessing

Implement the following steps to preprocess the dataset you created:

- convert the all reviews into the lower case.
- remove the HTML and URLs from the reviews
- remove non-alphabetical characters
- remove extra spaces
- perform contractions on the reviews, e.g., won't -> will not. Include as many contractions in English that you can think of.

You can either use Pandas package functions or any other built-in functions. Do not try to im plement the above processes manually. Most of the above processes can be performed with o ne line of code
In your report, print the average length of the reviews in terms of character length in your dat aset before and after cleaning (to be printed by .py file).

a. Average length of reviews before data cleaning: 322.619475
b. Average length of reviews after data cleaning: 301.630745
    - Str.lower() is used to convert all the characters to lowercase.

- To remove URL, we used regex which filters out every url embedded into the s tring. Beautiful Soap was used to eliminate HTML markups from the review bo dy
- Regex was used with substitution to remove all the non-alphabetic characters.
- Same for the multiple spaces we used regex with substitution.
- Used the function contraction which substitutes any word which is in contractio n dictionary to its root word.

3. PREPROCESSING

Use NLTK package to process your dataset:

- Remove the stopwords
- Perform lemmatization

Print three sample reviews before and after data cleaning + preprocessing. In the .py file, print the average length of the reviews in terms of character length in before and after pro cessing.

A:

3 Sample Reviews before data cleaning and data preprocessing

- I put these on a south facing window and parts of the shades disintigrated over time d ue to the intense sun where I live - Colorado. Hunter Douglas would not cover them with their warranty as it states that exposure to the elements voids the warranty. The y consider the sun an element...I guess you need to use these shades where the sun do esn't shine. These are way too expensive to be replacing them every several years. Ju st a heads up if you are considering buying these. I won't spend the money for these a gain. As a follow up- 8/29/07 - I wrote to the corporate office and Hunter Douglas is replacing the Sillouettes under warranty.
- The recipient loved her gift and that makes for a very satisfied customer. Thanks.
- Exactly what I expected when I ordered the product.

```
print("Sample Reviews before data cleaning \n")
for ele in review_sample:
    print(ele)
```

```
I put these on a south facing window and parts of the shades disintigrated over time due to the intense sun where I live - Colo
rado.  Hunter Douglas would not cover them with their warranty as it states that exposure to the elements voids the warranty.
They consider the sun an element...I guess you need to use these shades where the sun doesn't shine. These are way too expensiv
e to be replacing them every several years.  Just a heads up if you are considering buying these. I won't spend the money for t
hese again. As a follow up- 8/29/07 - I wrote to the corporate office and Hunter Douglas is replacing the Sillouettes under war
ranty.
The recipient loved her gift and that makes for a very satisfied customer.  Thanks.
Exactly what I expected when I ordered the product.
Sample Reviews after data cleaning and pre-processing:
```

3 Sample Reviews after data cleaning and data preprocessing

- put south facing window part shade disintigrated time due intense sun live colorado hunter douglas would cover warranty state exposure element void warranty consider sun shine way expensive replacing every several year head considering buying spen d money follow wrote corporate office hunter douglas replacing sillouettes warranty
- recipient loved gift make satisfied customer thanks
- exactly expected ordered product

```
print("Sample Reviews after data cleaning and pre-processing:\n")
for ele in review_sample_2:
    print(ele)
```

```
put south facing window part shade disintigrated time due intense sun live colorado hunter douglas would cover warranty state e
xposure element void warranty consider sun shine way expensive replacing every several year head considering buying spend money
follow wrote corporate office hunter douglas replacing sillouettes warranty
recipient loved gift make satisfied customer thanks
exactly expected ordered product
```

```
print("Average length of reviews before data cleaning: ", aclbdc)
print("Average length of reviews after data cleaning: ", acladc)

print("Average length of reviews before data pre-processing: ", aclbdp)
print("Average length of reviews after data pre-processing: ", acladp)
```

```
Average length of reviews before data cleaning:  322.619475
Average length of reviews after data cleaning:  301.630745
Average length of reviews before data pre-processing:  301.630745
Average length of reviews after data pre-processing:  184.227195
```

4. Feature Extraction

Use sklearn to extract TF-IDF features. At this point, you should have created a dataset which consists of features and binary labels for the reviews you selected.

A:

TF-IDF Vectorizer library was used to extract features from the reviews. Fit and transform was applied on the training dataset and only transform on testing dataset.

TF-IDF Feature Extraction

```
tfidf_vect = TfidfVectorizer(min_df = 0.001)
X_train = tfidf_vect.fit_transform(train['review_body'])
X_train = pd.DataFrame(X_train.toarray(), columns = tfidf_vect.get_feature_names())
X_test = tfidf_vect.transform(test['review_body'])
X_test = pd.DataFrame(X_test.toarray(), columns = tfidf_vect.get_feature_names())
Y_train = train['label']
Y_test = test['label']
```

5. Perceptron

Train a Perceptron model on your training dataset using the sklearn built-in implementation. Report Accuracy, Precision, Recall, and f1-score on both the training and testing split of your dataset. These 8 values should be printed in separate lines by the .py file.

A:

Perceptron was imported from the sklearn library with max iteration/epochs to 80, stopping criterion to 0.00001, random_state to 200 for consistency and learning rate at 0.01.

Accuracy, Precision, Recall and F1-score

Train: 0.87, 0.84, 0.91, 0.87
Test: 0.86, 0.84, 0.90, 0.87

Perceptron

```
def metrics(true, pred):
    tn, fp, fn, tp = cm(true, pred).ravel()
    acc = (tp + tn)/(tn + fp + fn + tp)
    prec = tp/(tp + fp)
    rec = tp / (tp + fn)
    f1 = 2*(rec * prec) / (rec + prec)
    return [acc, prec, rec, f1]

def print_seq(score_list):
    print("Accuracy: %.2f" % score_list[0])
    print("Precision: %.2f" % score_list[1])
    print("Recall: %.2f" % score_list[2])
    print("F1-score %.2f" % score_list[3])

percept = Perceptron(max_iter = 80, tol = 1e-5, verbose = 1, random_state = 200, eta0 = 0.01)
percept.fit(X_train, Y_train)
Y_train_pred = percept.predict(X_train)
train_score = metrics(Y_train, Y_train_pred)

# Predicting on test dataset
Y_test_pred = percept.predict(X_test)
test_score = metrics(Y_test, Y_test_pred)
print(train_score)
print(test_score)
# print("TRAIN_SCORES:")
# print_seq(train_score)
# print("TEST_SCORES:")
# print_seq(test_score)
```

```
-- Epoch 1
Norm: 0.72, NNZs: 2649, Bias: -0.010000, T: 160000, Avg. loss: 0.001509
Total training time: 0.43 seconds.
-- Epoch 2
Norm: 0.77, NNZs: 2649, Bias: 0.000000, T: 320000, Avg. loss: 0.001524
Total training time: 0.87 seconds.
-- Epoch 3
Norm: 0.79, NNZs: 2649, Bias: 0.010000, T: 480000, Avg. loss: 0.001518
Total training time: 1.29 seconds.
-- Epoch 4
Norm: 0.82, NNZs: 2649, Bias: 0.000000, T: 640000, Avg. loss: 0.001529
Total training time: 1.72 seconds.
-- Epoch 5
Norm: 0.82, NNZs: 2649, Bias: -0.010000, T: 800000, Avg. loss: 0.001540
Total training time: 2.16 seconds.
-- Epoch 6
Norm: 0.82, NNZs: 2649, Bias: 0.000000, T: 960000, Avg. loss: 0.001535
Total training time: 2.58 seconds.
Convergence after 6 epochs took 2.58 seconds
[0.87166875, 0.844593103608469, 0.9104589728822896, 0.8762900883857401]
[0.864525, 0.8400462427745665, 0.9026135347311934, 0.87020670163588 9]
```

6. SVM

Train an SVM model on your training dataset using the sklearn built-in implementation. Report Accuracy, Precision, Recall, and f1-score on both the training and testing split of your dataset. These 8 values should be printed in separate lines by the .py file.

A:

Linear SVM was used with max iteration at 5000. Fit method was used to train the SVM model on the training dataset and predict on testing dataset.

Accuracy, Precision, Recall and F1-score:

Train: 0.89, 0.90, 0.89, 0.89
Test: 0.89, 0.89, 0.88, 0.89

SVM

```
svc = SVC(max_iter = 5000)
model_svc = svc.fit(X_train, Y_train)
pred_train_svm = model_svc.predict(X_train)
pred_test_svm = model_svc.predict(X_test)
train_svm_score = metrics(Y_train, pred_train_svm)
test_svm_score = metrics(Y_test, pred_test_svm)

print(train_svm_score)
print(test_svm_score)
# print("TRAIN_SCORES:")
# print_seq(train_svm_score)
# print("TEST_SCORES:")
# print_seq(test_svm_score)
```

```
[0.89858125, 0.902243569487455, 0.8936700302977189, 0.8979363352181597]
[0.89065, 0.8972561283163523, 0.883881546258571, 0.8905186223468162]
```

7. Logistic Regression

Train a Logistic Regression model on your training dataset using the sklearn built-in implementation. Report Accuracy, Precision, Recall, and f1-score on both the training and testing split of your dataset. These 8 values should be printed in separate lines by the .py file.

A:

In Logistic Regression we used max iteration to 1000, penalty to l2 and solver to libline ar.

Accuracy, Recall, Precision and F1-score:

Train: 0.89, 0.90, 0.89, 0.89
Test: 0.89, 0.89, 0.88, 0.89

Logistic Regression

```
log_reg = LR(verbose=1, solver='liblinear',random_state=0, C=5, penalty='l2',max_iter=1000)
model = log_reg.fit(X_train, Y_train)
pred_train_LR = model.predict(X_train)
pred_test_LR = model.predict(X_test)
train_LR_score = metrics(Y_train, pred_train_LR)
test_LR_score = metrics(Y_test, pred_test_LR)

print(train_LR_score)
print(test_LR_score)
# print("TRAIN_SCORES:")
# print_seq(train_LR_score)
# print("TEST_SCORES:")
# print_seq(test_LR_score)
```

```
[LibLinear][0.89938125, 0.9028386624050937, 0.8947342063750406, 0.898768164697449]
[0.890725, 0.8969113719957676, 0.884477789923482, 0.8906511895529482]
```

8. Naïve Bayes

Train a Multinomial Naive Bayes model on your training dataset using the sklearn built-in implementation. Report Accuracy, Precision, Recall, and f1-score on both the training and testing split of your dataset. These 8 values should be printed in separate lines by the .py f ile.

A:

Accuracy, Precision, Recall and F1-score

Train: 0.86, 0.86, 0.86, 0.86
Test: 0.86, 0.86, 0.86, 0.86

Naive Bayes

```
MNB = MultinomialNB()
MNB.fit(X_train, Y_train)
pred_train_MNB = MNB.predict(X_train)
pred_test_MNB = MNB.predict(X_test)
train_MNB_score = metrics(Y_train, pred_train_MNB)
test_MNB_score = metrics(Y_test, pred_test_MNB)

print(train_MNB_score)
print(test_MNB_score)
# print("TRAIN_SCORES:")
# print_seq(train_MNB_score)
# print("TEST_SCORES:")
# print_seq(test_MNB_score)
```

```
[0.86630625, 0.8647755198782481, 0.8679044495079751, 0.8663371595318583]
[0.862325, 0.8630967165068799, 0.8633111398191394, 0.8632039148471073]
```

HW1-CSCI544

September 7, 2021

[1]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import contractions
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn.metrics import confusion_matrix as cm
from sklearn.svm import LinearSVC as SVC
from sklearn.linear_model import LogisticRegression as LR
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression as LR
import warnings
warnings.filterwarnings('ignore')
```

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

[2]:
```
#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

[3]:
```
df = pd.read_csv("../amazon_reviews_us_Kitchen_v1_00.tsv", sep = "\t",␣
 ↪error_bad_lines=False, warn_bad_lines=False)
```

0.2 Keep Reviews and Ratings

[4]:
```
df = df[['star_rating', 'review_body']]
print("REVIEWS SAMPPLE WITH RATING")
print(df.head(3))
print("STATISIICS OF THE REVIEWS: ")
print("rating 1: ", len(df[df.star_rating == 1]))
print("rating 2: ", len(df[df.star_rating == 2]))
print("rating 3: ", len(df[df.star_rating == 3]))
print("rating 4: ", len(df[df.star_rating == 4]))
print("rating 5: ", len(df[df.star_rating == 5]))
df = df.dropna()
```

```
REVIEWS SAMPPLE WITH RATING
   star_rating                                        review_body
0          5.0                Beautiful.  Looks great on counter.
1          5.0  I personally have 5 days sets and have also bo…
2          5.0  Fabulous and worth every penny. Used for clean…
STATISIICS OF THE REVIEWS:
rating 1:  426900
rating 2:  241948
rating 3:  349547
rating 4:  731733
rating 5:  3124759
```

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'

[5]:
```
df['label'] = df.apply(lambda row: 1 if row.star_rating > 3 else 0, axis = 1)
r0 = len(df[df.label == 0])
r1 = len(df[df.label == 1])
r3 = len(df[df.star_rating == 3])
df = df[df.star_rating != 3]
print(r0, ", ", r1, ", ", r3)
```

```
1018348 ,  3856296 ,  349539
```

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

[6]:
```
positive = df[df['label'] == 1]
negative = df[df['label'] == 0]
positive = positive.sample(n = 100000, random_state = 200)
negative = negative.sample(n = 100000, random_state=200)
dataset = pd.concat([positive, negative])
dataset = dataset.reset_index(drop = True)
train=dataset.sample(frac=0.8,random_state=200)
test = dataset.drop(train.index)
```

```
train = train.reset_index(drop = True)
test = test.reset_index(drop = True)
```

2 Data Cleaning

2.1 Convert the all reviews into the lower case.

[7]:
```
aclbdc = (sum(train.review_body.str.len()) + sum(test.review_body.str.len()))/
↪200000
review_sample = list(train.review_body.head(3))
# convert all the reviews to lowercase
train['review_body'] = train['review_body'].str.lower()
test['review_body'] = test['review_body'].str.lower()
```

2.2 remove the HTML and URLs from the reviews

[8]:
```
def remove_html_and_url(x):
    x = re.sub(r'(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)', '',␣
↪x, flags=re.MULTILINE)
    soup = BeautifulSoup(x, 'html.parser')
    x = soup.get_text()
    return x

train.review_body = train.review_body.apply(lambda x: remove_html_and_url(x))
test.review_body = test.review_body.apply(lambda x: remove_html_and_url(x))
```

2.3 remove non-alphabetical characters

[9]:
```
train.review_body = train.review_body.apply(lambda x: re.sub("[^a-zA-Z']+", "␣
↪", x))
test.review_body = test.review_body.apply(lambda x: re.sub("[^a-zA-Z']+", " ",␣
↪x))
```

2.4 Remove the extra spaces between the words

[10]:
```
train.review_body = train.review_body.apply(lambda x: re.sub(' +', ' ', x))
test.review_body = test.review_body.apply(lambda x: re.sub(' +', ' ', x))
```

2.5 perform contractions on the reviews.

[11]:
```
def contractionfunction(s):
    s = contractions.fix(s)
    s = re.sub("[^a-zA-Z]+", " ", s)
    return s
train.review_body = train.review_body.apply(lambda x: contractionfunction(x))
test.review_body = test.review_body.apply(lambda x: contractionfunction(x))
```

```
acladc = (sum(train.review_body.str.len()) + sum(test.review_body.str.len()))/
↪200000
```

3 Pre-processing

3.1 remove the stop words

[12]:
```
aclbdp = (sum(train.review_body.str.len()) + sum(test.review_body.str.len()))/
↪200000
def remove_stopwords(s):
    text_tokens = word_tokenize(s)
    tokens_without_sw = [word for word in text_tokens if not word in stopwords.
↪words('english')]
    return tokens_without_sw
train.review_body = train.review_body.apply(remove_stopwords)
test.review_body = test.review_body.apply(remove_stopwords)
print(train.review_body)
```

```
0         [put, south, facing, window, parts, shades, di…
1         [recipient, loved, gift, makes, satisfied, cus…
2                     [exactly, expected, ordered, product]
3                                                   [love]
4         [handy, gadget, monitor, temps, fridge, differ…
                               …
159995                          [vegetable, peeler, thing]
159996    [pans, best, ever, used, gave, little, cooking…
159997    [boyfriend, moved, new, apartment, needed, dis…
159998                                              [stars]
159999    [worked, days, ring, tightens, bottle, snapped…
Name: review_body, Length: 160000, dtype: object
```

3.2 perform lemmatization

[13]:
```
lemmatizer = WordNetLemmatizer()
def lemmatize(x):
    lemmatize_tokens = [lemmatizer.lemmatize(word) for word in x]
    x = ' '.join(lemmatize_tokens)
    return x

train.review_body = train.review_body.apply(lemmatize)
test.review_body = test.review_body.apply(lemmatize)
review_sample_2 = list(train.review_body.head(3))
acladp = (sum(train.review_body.str.len()) + sum(test.review_body.str.len()))/
↪200000

print("STATISTICS OF THREE CLASES: ")
print("label 0: ", r0, ", label 1: ", r1, ", label 3: ", r3)
```

```
print("Average length of reviews before data cleaning: ", aclbdc)
print("Average length of reviews after data cleaning: ", acladc)

print("Average length of reviews before data pre-processing: ", aclbdp)
print("Average length of reviews after data pre-processing: ", acladp)

print("Sample Reviews before data cleaning \n")
for ele in review_sample:
    print(ele)
print("Sample Reviews after data cleaning and pre-processing:\n")
for ele in review_sample_2:
    print(ele)
```

```
STATISTICS OF THREE CLASES:
label 0:  1018348 , label 1:  3856296 , label 3:  349539
Average length of reviews before data cleaning:  322.619475
Average length of reviews after data cleaning:  301.630745
Average length of reviews before data pre-processing:  301.630745
Average length of reviews after data pre-processing:  184.227195
Sample Reviews before data cleaning

I put these on a south facing window and parts of the shades disintigrated over
time due to the intense sun where I live - Colorado.  Hunter Douglas would not
cover them with their warranty as it states that exposure to the elements voids
the warranty.  They consider the sun an element…I guess you need to use these
shades where the sun doesn't shine. These are way too expensive to be replacing
them every several years.  Just a heads up if you are considering buying these.
I won't spend the money for these again. As a follow up- 8/29/07 - I wrote to
the corporate office and Hunter Douglas is replacing the Sillouettes under
warranty.
The recipient loved her gift and that makes for a very satisfied customer.
Thanks.
Exactly what I expected when I ordered the product.
Sample Reviews after data cleaning and pre-processing:

put south facing window part shade disintigrated time due intense sun live
colorado hunter douglas would cover warranty state exposure element void
warranty consider sun shine way expensive replacing every several year head
considering buying spend money follow wrote corporate office hunter douglas
replacing sillouettes warranty
recipient loved gift make satisfied customer thanks
exactly expected ordered product
```

4 TF-IDF Feature Extraction

[14]:
```
tfidf_vect = TfidfVectorizer(min_df = 0.001)
X_train = tfidf_vect.fit_transform(train['review_body'])
X_train = pd.DataFrame(X_train.toarray(), columns = tfidf_vect.
 ↪get_feature_names())
X_test = tfidf_vect.transform(test['review_body'])
X_test = pd.DataFrame(X_test.toarray(), columns = tfidf_vect.
 ↪get_feature_names())
Y_train = train['label']
Y_test = test['label']
```

5 Perceptron

[15]:
```
def metrics(true, pred):
    tn, fp, fn, tp = cm(true, pred).ravel()
    acc = (tp + tn)/(tn + fp + fn + tp)
    prec = tp/(tp + fp)
    rec = tp / (tp + fn)
    f1 = 2*(rec * prec) / (rec + prec)
    return [acc, prec, rec, f1]

def print_seq(score_list):
    print("Accuracy: %.2f" % score_list[0])
    print("Precision: %.2f" % score_list[1])
    print("Recall: %.2f" % score_list[2])
    print("F1-score %.2f" % score_list[3])

percept = Perceptron(max_iter = 80, tol = 1e-5, verbose = 1, random_state =␣
 ↪200, eta0 = 0.01)
percept.fit(X_train, Y_train)
Y_train_pred = percept.predict(X_train)
train_score = metrics(Y_train, Y_train_pred)

# Predicting on test dataset
Y_test_pred = percept.predict(X_test)
test_score = metrics(Y_test, Y_test_pred)
print(train_score)
print(test_score)
# print("TRAIN_SCORES:")
# print_seq(train_score)
# print("TEST_SCORES:")
# print_seq(test_score)
```

```
-- Epoch 1
Norm: 0.72, NNZs: 2649, Bias: -0.010000, T: 160000, Avg. loss: 0.001509
Total training time: 0.45 seconds.
```

```
-- Epoch 2
Norm: 0.77, NNZs: 2649, Bias: 0.000000, T: 320000, Avg. loss: 0.001524
Total training time: 0.85 seconds.
-- Epoch 3
Norm: 0.79, NNZs: 2649, Bias: 0.010000, T: 480000, Avg. loss: 0.001518
Total training time: 1.25 seconds.
-- Epoch 4
Norm: 0.82, NNZs: 2649, Bias: 0.000000, T: 640000, Avg. loss: 0.001529
Total training time: 1.64 seconds.
-- Epoch 5
Norm: 0.82, NNZs: 2649, Bias: -0.010000, T: 800000, Avg. loss: 0.001540
Total training time: 2.05 seconds.
-- Epoch 6
Norm: 0.82, NNZs: 2649, Bias: 0.000000, T: 960000, Avg. loss: 0.001535
Total training time: 2.45 seconds.
Convergence after 6 epochs took 2.45 seconds
[0.87166875, 0.844593103608469, 0.9104589728822896, 0.8762900883857401]
[0.864525, 0.8400462427745665, 0.9026135347311934, 0.870206701635889]
```

6 SVM

[16]:
```
svc = SVC(max_iter = 5000)
model_svc = svc.fit(X_train, Y_train)
pred_train_svm = model_svc.predict(X_train)
pred_test_svm = model_svc.predict(X_test)
train_svm_score = metrics(Y_train, pred_train_svm)
test_svm_score = metrics(Y_test, pred_test_svm)

print(train_svm_score)
print(test_svm_score)
# print("TRAIN_SCORES:")
# print_seq(train_sum_score)
# print("TEST_SCORES:")
# print_seq(test_sum_score)
```

```
[0.89858125, 0.902243569487455, 0.8936700302977189, 0.8979363352181597]
[0.89065, 0.8972561283163523, 0.883881546258571, 0.8905186223468162]
```

7 Logistic Regression

[17]:
```
log_reg = LR(verbose=1, solver='liblinear',random_state=0, C=5,␣
 ↪penalty='l2',max_iter=1000)
model = log_reg.fit(X_train, Y_train)
pred_train_LR = model.predict(X_train)
pred_test_LR = model.predict(X_test)
train_LR_score = metrics(Y_train, pred_train_LR)
```

```
test_LR_score = metrics(Y_test, pred_test_LR)

print(train_LR_score)
print(test_LR_score)
# print("TRAIN_SCORES:")
# print_seq(train_LR_score)
# print("TEST_SCORES:")
# print_seq(test_LR_score)
```

```
[LibLinear][0.89938125, 0.9028386624050937, 0.8947342063750406,
0.898768164697449]
[0.890725, 0.8969113719957676, 0.884477789923482, 0.8906511895529482]
```

8 Naive Bayes

[18]:
```
MNB = MultinomialNB()
MNB.fit(X_train, Y_train)
pred_train_MNB = MNB.predict(X_train)
pred_test_MNB = MNB.predict(X_test)
train_MNB_score = metrics(Y_train, pred_train_MNB)
test_MNB_score = metrics(Y_test, pred_test_MNB)

print(train_MNB_score)
print(test_MNB_score)
# print("TRAIN_SCORES:")
# print_seq(train_MNB_score)
# print("TEST_SCORES:")
# print_seq(test_MNB_score)
```

```
[0.86630625, 0.8647755198782481, 0.8679044495079751, 0.8663371595318583]
[0.862325, 0.8630967165068799, 0.8633111398191394, 0.8632039148471073]
```

[ ]:
```

```