HW1-CSCI544

September 9, 2021

[238]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
import re
from bs4 import BeautifulSoup
import contractions
import warnings
warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/shreenidhihegde/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/shreenidhihegde/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[61]:
```
#! 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

[239]:
```
 """
 Here we are reading the data from tsv file as pandas dataframe.
 We use '\t' to seperate columns by a tab and "error_bad_lines = False" to drop␣
 ↪bad lines from the DataFrame

 """
text_data = pd.read_csv("data.tsv",error_bad_lines = False, sep = '\t')
```

```
/Users/shreenidhihegde/opt/anaconda3/envs/NLP/lib/python3.9/site-
packages/IPython/core/interactiveshell.py:3441: FutureWarning: The
error_bad_lines argument has been deprecated and will be removed in a future
version.
```

```
  exec(code_obj, self.user_global_ns, self.user_ns)
b'Skipping line 16148: expected 15 fields, saw 22\nSkipping line 20100: expected
15 fields, saw 22\nSkipping line 45178: expected 15 fields, saw 22\nSkipping
line 48700: expected 15 fields, saw 22\nSkipping line 63331: expected 15 fields,
saw 22\n'
b'Skipping line 86053: expected 15 fields, saw 22\nSkipping line 88858: expected
15 fields, saw 22\nSkipping line 115017: expected 15 fields, saw 22\n'
b'Skipping line 137366: expected 15 fields, saw 22\nSkipping line 139110:
expected 15 fields, saw 22\nSkipping line 165540: expected 15 fields, saw
22\nSkipping line 171813: expected 15 fields, saw 22\n'
b'Skipping line 203723: expected 15 fields, saw 22\nSkipping line 209366:
expected 15 fields, saw 22\nSkipping line 211310: expected 15 fields, saw
22\nSkipping line 246351: expected 15 fields, saw 22\nSkipping line 252364:
expected 15 fields, saw 22\n'
b'Skipping line 267003: expected 15 fields, saw 22\nSkipping line 268957:
expected 15 fields, saw 22\nSkipping line 303336: expected 15 fields, saw
22\nSkipping line 306021: expected 15 fields, saw 22\nSkipping line 311569:
expected 15 fields, saw 22\nSkipping line 316767: expected 15 fields, saw
22\nSkipping line 324009: expected 15 fields, saw 22\n'
b'Skipping line 359107: expected 15 fields, saw 22\nSkipping line 368367:
expected 15 fields, saw 22\nSkipping line 381180: expected 15 fields, saw
22\nSkipping line 390453: expected 15 fields, saw 22\n'
b'Skipping line 412243: expected 15 fields, saw 22\nSkipping line 419342:
expected 15 fields, saw 22\nSkipping line 457388: expected 15 fields, saw 22\n'
b'Skipping line 459935: expected 15 fields, saw 22\nSkipping line 460167:
expected 15 fields, saw 22\nSkipping line 466460: expected 15 fields, saw
22\nSkipping line 500314: expected 15 fields, saw 22\nSkipping line 500339:
expected 15 fields, saw 22\nSkipping line 505396: expected 15 fields, saw
22\nSkipping line 507760: expected 15 fields, saw 22\nSkipping line 513626:
expected 15 fields, saw 22\n'
b'Skipping line 527638: expected 15 fields, saw 22\nSkipping line 534209:
expected 15 fields, saw 22\nSkipping line 535687: expected 15 fields, saw
22\nSkipping line 547671: expected 15 fields, saw 22\nSkipping line 549054:
expected 15 fields, saw 22\n'
b'Skipping line 599929: expected 15 fields, saw 22\nSkipping line 604776:
expected 15 fields, saw 22\nSkipping line 609937: expected 15 fields, saw
22\nSkipping line 632059: expected 15 fields, saw 22\nSkipping line 638546:
expected 15 fields, saw 22\n'
b'Skipping line 665017: expected 15 fields, saw 22\nSkipping line 677680:
expected 15 fields, saw 22\nSkipping line 684370: expected 15 fields, saw
22\nSkipping line 720217: expected 15 fields, saw 29\n'
b'Skipping line 723240: expected 15 fields, saw 22\nSkipping line 723433:
expected 15 fields, saw 22\nSkipping line 763891: expected 15 fields, saw 22\n'
b'Skipping line 800288: expected 15 fields, saw 22\nSkipping line 802942:
expected 15 fields, saw 22\nSkipping line 803379: expected 15 fields, saw
22\nSkipping line 805122: expected 15 fields, saw 22\nSkipping line 821899:
expected 15 fields, saw 22\nSkipping line 831707: expected 15 fields, saw
```

```
22\nSkipping line 842829: expected 15 fields, saw 22\nSkipping line 843604:
expected 15 fields, saw 22\n'
b'Skipping line 863904: expected 15 fields, saw 22\nSkipping line 875655:
expected 15 fields, saw 22\nSkipping line 886796: expected 15 fields, saw
22\nSkipping line 892299: expected 15 fields, saw 22\nSkipping line 902518:
expected 15 fields, saw 22\nSkipping line 903079: expected 15 fields, saw
22\nSkipping line 912678: expected 15 fields, saw 22\n'
b'Skipping line 932953: expected 15 fields, saw 22\nSkipping line 936838:
expected 15 fields, saw 22\nSkipping line 937177: expected 15 fields, saw
22\nSkipping line 947695: expected 15 fields, saw 22\nSkipping line 960713:
expected 15 fields, saw 22\nSkipping line 965225: expected 15 fields, saw
22\nSkipping line 980776: expected 15 fields, saw 22\n'
b'Skipping line 999318: expected 15 fields, saw 22\nSkipping line 1007247:
expected 15 fields, saw 22\nSkipping line 1015987: expected 15 fields, saw
22\nSkipping line 1018984: expected 15 fields, saw 22\nSkipping line 1028671:
expected 15 fields, saw 22\n'
b'Skipping line 1063360: expected 15 fields, saw 22\nSkipping line 1066195:
expected 15 fields, saw 22\nSkipping line 1066578: expected 15 fields, saw
22\nSkipping line 1066869: expected 15 fields, saw 22\nSkipping line 1068809:
expected 15 fields, saw 22\nSkipping line 1069505: expected 15 fields, saw
22\nSkipping line 1087983: expected 15 fields, saw 22\nSkipping line 1108184:
expected 15 fields, saw 22\n'
b'Skipping line 1118137: expected 15 fields, saw 22\nSkipping line 1142723:
expected 15 fields, saw 22\nSkipping line 1152492: expected 15 fields, saw
22\nSkipping line 1156947: expected 15 fields, saw 22\nSkipping line 1172563:
expected 15 fields, saw 22\n'
b'Skipping line 1209254: expected 15 fields, saw 22\nSkipping line 1212966:
expected 15 fields, saw 22\nSkipping line 1236533: expected 15 fields, saw
22\nSkipping line 1237598: expected 15 fields, saw 22\n'
b'Skipping line 1273825: expected 15 fields, saw 22\nSkipping line 1277898:
expected 15 fields, saw 22\nSkipping line 1283654: expected 15 fields, saw
22\nSkipping line 1286023: expected 15 fields, saw 22\nSkipping line 1302038:
expected 15 fields, saw 22\nSkipping line 1305179: expected 15 fields, saw 22\n'
b'Skipping line 1326022: expected 15 fields, saw 22\nSkipping line 1338120:
expected 15 fields, saw 22\nSkipping line 1338503: expected 15 fields, saw
22\nSkipping line 1338849: expected 15 fields, saw 22\nSkipping line 1341513:
expected 15 fields, saw 22\nSkipping line 1346493: expected 15 fields, saw
22\nSkipping line 1373127: expected 15 fields, saw 22\n'
b'Skipping line 1389508: expected 15 fields, saw 22\nSkipping line 1413951:
expected 15 fields, saw 22\nSkipping line 1433626: expected 15 fields, saw 22\n'
b'Skipping line 1442698: expected 15 fields, saw 22\nSkipping line 1472982:
expected 15 fields, saw 22\nSkipping line 1482282: expected 15 fields, saw
22\nSkipping line 1487808: expected 15 fields, saw 22\nSkipping line 1500636:
expected 15 fields, saw 22\n'
b'Skipping line 1511479: expected 15 fields, saw 22\nSkipping line 1532302:
expected 15 fields, saw 22\nSkipping line 1537952: expected 15 fields, saw
22\nSkipping line 1539951: expected 15 fields, saw 22\nSkipping line 1541020:
expected 15 fields, saw 22\n'
```

```
b'Skipping line 1594217: expected 15 fields, saw 22\nSkipping line 1612264:
expected 15 fields, saw 22\nSkipping line 1615907: expected 15 fields, saw
22\nSkipping line 1621859: expected 15 fields, saw 22\n'
b'Skipping line 1653542: expected 15 fields, saw 22\nSkipping line 1671537:
expected 15 fields, saw 22\nSkipping line 1672879: expected 15 fields, saw
22\nSkipping line 1674523: expected 15 fields, saw 22\nSkipping line 1677355:
expected 15 fields, saw 22\nSkipping line 1703907: expected 15 fields, saw 22\n'
b'Skipping line 1713046: expected 15 fields, saw 22\nSkipping line 1722982:
expected 15 fields, saw 22\nSkipping line 1727290: expected 15 fields, saw
22\nSkipping line 1744482: expected 15 fields, saw 22\n'
b'Skipping line 1803858: expected 15 fields, saw 22\nSkipping line 1810069:
expected 15 fields, saw 22\nSkipping line 1829751: expected 15 fields, saw
22\nSkipping line 1831699: expected 15 fields, saw 22\n'
b'Skipping line 1863131: expected 15 fields, saw 22\nSkipping line 1867917:
expected 15 fields, saw 22\nSkipping line 1874790: expected 15 fields, saw
22\nSkipping line 1879952: expected 15 fields, saw 22\nSkipping line 1880501:
expected 15 fields, saw 22\nSkipping line 1886655: expected 15 fields, saw
22\nSkipping line 1887888: expected 15 fields, saw 22\nSkipping line 1894286:
expected 15 fields, saw 22\nSkipping line 1895400: expected 15 fields, saw 22\n'
b'Skipping line 1904040: expected 15 fields, saw 22\nSkipping line 1907604:
expected 15 fields, saw 22\nSkipping line 1915739: expected 15 fields, saw
22\nSkipping line 1921514: expected 15 fields, saw 22\nSkipping line 1939428:
expected 15 fields, saw 22\nSkipping line 1944342: expected 15 fields, saw
22\nSkipping line 1949699: expected 15 fields, saw 22\nSkipping line 1961872:
expected 15 fields, saw 22\n'
b'Skipping line 1968846: expected 15 fields, saw 22\nSkipping line 1999941:
expected 15 fields, saw 22\nSkipping line 2001492: expected 15 fields, saw
22\nSkipping line 2011204: expected 15 fields, saw 22\nSkipping line 2025295:
expected 15 fields, saw 22\n'
b'Skipping line 2041266: expected 15 fields, saw 22\nSkipping line 2073314:
expected 15 fields, saw 22\nSkipping line 2080133: expected 15 fields, saw
22\nSkipping line 2088521: expected 15 fields, saw 22\n'
b'Skipping line 2103490: expected 15 fields, saw 22\nSkipping line 2115278:
expected 15 fields, saw 22\nSkipping line 2153174: expected 15 fields, saw
22\nSkipping line 2161731: expected 15 fields, saw 22\n'
b'Skipping line 2165250: expected 15 fields, saw 22\nSkipping line 2175132:
expected 15 fields, saw 22\nSkipping line 2206817: expected 15 fields, saw
22\nSkipping line 2215848: expected 15 fields, saw 22\nSkipping line 2223811:
expected 15 fields, saw 22\n'
b'Skipping line 2257265: expected 15 fields, saw 22\nSkipping line 2259163:
expected 15 fields, saw 22\nSkipping line 2263291: expected 15 fields, saw 22\n'
b'Skipping line 2301943: expected 15 fields, saw 22\nSkipping line 2304371:
expected 15 fields, saw 22\nSkipping line 2306015: expected 15 fields, saw
22\nSkipping line 2312186: expected 15 fields, saw 22\nSkipping line 2314740:
expected 15 fields, saw 22\nSkipping line 2317754: expected 15 fields, saw 22\n'
b'Skipping line 2383514: expected 15 fields, saw 22\n'
b'Skipping line 2449763: expected 15 fields, saw 22\n'
b'Skipping line 2589323: expected 15 fields, saw 22\n'
```

```
b'Skipping line 2775036: expected 15 fields, saw 22\n'
b'Skipping line 2935174: expected 15 fields, saw 22\n'
b'Skipping line 3078830: expected 15 fields, saw 22\n'
b'Skipping line 3123091: expected 15 fields, saw 22\n'
b'Skipping line 3185533: expected 15 fields, saw 22\n'
b'Skipping line 4150395: expected 15 fields, saw 22\n'
b'Skipping line 4748401: expected 15 fields, saw 22\n'
```

0.2 Keep Reviews and Ratings

[240]:

```
#We are using dropna to look for missing values in the rows which has no␣
↪review body and drop the corresponding rows.

text_data.dropna(subset = ['review_body'], inplace= True)
text_data.dropna(subset = ['star_rating'], inplace= True)


# We Keep only the reviews and ratings of the initial data
text_data = text_data[["star_rating","review_body"]]

# We are including 3 sample reviews with the corresponding rating
print ("\n -x--x- Three sample reviews with corresponding ratings -x--x-␣
↪\n")
print (text_data.sample(n=3, random_state=100))

# We are reporting statistics of the ratings
star_counts = text_data["star_rating"].value_counts()
print("\n -x--x- Statistics of the ratings -x--x- \n")
print(star_counts)
```

```
 -x--x- Three sample reviews with corresponding ratings -x--x-

         star_rating                                       review_body
2264076          5.0  Do you know what's better than a Seattle Seaha…
2973343          4.0  the soup bowls are little bit smaller for nood…
2090625          3.0   They stain easy & handles get really hot to grab

 -x--x- Statistics of the ratings -x--x-

5.0    3124595
4.0     731701
1.0     426870
3.0     349539
2.0     241939
Name: star_rating, dtype: int64
```

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'

[241]:
```
# Using np.where we are putting conditions to label the ratings >=4 as 1 and␣
↪ratins <=2 as 0.
#The remainign ratings which is labelled as 3 will be labelled as -1.

text_data['label'] = np.where(text_data["star_rating"] >=4,1, np.
↪where(text_data["star_rating"] <= 2,0,-1))
rating_count = text_data['label'].value_counts()

#We are getting the counts of all review labels before removing the reviews␣
↪with rating 3
print("\n -x--x- Counts of all review labels before removing the reviews with␣
↪rating 3 -x--x-\n")
print(rating_count)

#Discarding the reviews with rating 3
text_data = text_data[text_data["label"]!= -1]

#We are getting the counts of all review labels after removing the reviews with␣
↪rating 3
print("\n -x--x- Counts of all review labels after removing the reviews with␣
↪rating 3 -x--x- \n")
print(text_data['label'].value_counts())
```

```
 -x--x- Counts of all review labels before removing the reviews with rating 3 -x
--x-

 1    3856296
 0     668809
-1     349539
Name: label, dtype: int64

 -x--x- Counts of all review labels after removing the reviews with rating 3 -x
--x-

1    3856296
0     668809
Name: label, dtype: int64
```

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

[242]:
```
#We select sample of 100,000 positive and 100,000 negative reviews
negative = text_data.label[text_data.label.eq(0)].sample(100000).index
```

```
positive = text_data.label[text_data.label.eq(1)].sample(100000).index

#We combine both the selected samples
text_data = text_data.loc[negative.union(positive)]

# We are getting mean of each row based on characters in each row and then␣
↪finding the mean of all the rows
review_mean = text_data.review_body.apply(lambda x : len(str(x))).mean()
# print 3 sample reviews

print(f'\n -x--x- The average length of the reviews in terms of character␣
↪length in the dataset before cleaning is -x--x- \n{review_mean:.2f}')
print("\n -x--x- The sample reviews before cleaning -x--x- \n")
print(text_data["review_body"].sample(n=3, random_state=100))
```

```
 -x--x- The average length of the reviews in terms of character length in the
dataset before cleaning is -x--x-
323.45

 -x--x- The sample reviews before cleaning -x--x-

4471705    I ordered this Coffee Maker Nov 2009 and it wa…
231497     Cracked plastic base and is not airtight aroun…
1986586    This item was received with scratches all over…
Name: review_body, dtype: object
```

2 Data Cleaning

2.1 Convert the all reviews into the lower case.

[243]:
```
#We use str.lower() to convert all the characters into lower characters
text_data["review_body"] = text_data["review_body"].str.lower()
```

Remove the HTML and URLs from the reviews

[244]:
```
#We use Beatiful soup to remove all the HTML Tags from the dataframe
text_data["review_body"] = text_data["review_body"].apply(lambda x:␣
↪BeautifulSoup(str(x),"html.parser").get_text())

# We are here removing the URLs from the reviews
Url_pattern = r'\s*(https?://|www\.)+\S+(\s+|$)'
text_data["review_body"] = text_data["review_body"].apply(lambda x: re.
↪sub(Url_pattern, " ", str(x), flags=re.UNICODE))
```

```
/Users/shreenidhihegde/opt/anaconda3/envs/NLP/lib/python3.9/site-
packages/bs4/__init__.py:417: MarkupResemblesLocatorWarning:
"https://www.amazon.com/gp/profile/a3fci1qhe8fngz" looks like a URL. Beautiful
Soup is not an HTTP client. You should probably use an HTTP client like requests
to get the document behind the URL, and feed that document to Beautiful Soup.
  warnings.warn(
```

2.2 remove non-alphabetical characters

[245]:
```
#First we remove all the words starts with digits
text_data["review_body"] = text_data["review_body"].apply(lambda x: re.
↪sub(r"[^\D']+", " ", str(x), flags=re.UNICODE))

#Next we remove all the words which starts with non alphabetic characters
text_data["review_body"] = text_data["review_body"].apply(lambda x: re.
↪sub(r"[^\w']+", " ", str(x), flags=re.UNICODE))
```

2.3 Remove the extra spaces between the words

[246]:
```
#We remove the extra spaces from the dataset
# text_data["review_body"] = text_data["review_body"].replace('\s+', ' ',␣
↪regex=True)
text_data["review_body"] = text_data["review_body"].apply(lambda x: re.
↪sub(r"\s+", " ", str(x), flags=re.UNICODE))
```

2.4 perform contractions on the reviews.

[247]:
```
#We perform contractions using contractions.fix

def contractionfunction(i):
    i = i.apply(lambda x: contractions.fix(x))
    return i

text_data["review_body"] = contractionfunction(text_data["review_body"])
#We convert the characters into lowercase again as after contractions some␣
↪words will get capitalized Ex: "i'm will" become "I am" after contraction.
text_data["review_body"] = text_data["review_body"].str.lower()
```

[248]:
```
review_mean_new = text_data.review_body.apply(lambda x :len(str(x))).mean()
print("\n -x--x- The average length of the reviews in terms of character length␣
↪in the dataset after cleaning is -x--x-\n", review_mean_new)
print("\n -x--x- The sample reviews after cleaning -x--x-\n")
print(text_data["review_body"].sample(n=3, random_state=100))
```

```
 -x--x- The average length of the reviews in terms of character length in the
dataset after cleaning is -x--x-
```

```
 310.22597

 -x--x- The sample reviews after cleaning -x--x-

4471705    i ordered this coffee maker nov and it was fin…
231497     cracked plastic base and is not airtight aroun…
1986586    this item was received with scratches all over…
Name: review_body, dtype: object
```

3 Pre-processing

3.1 remove the stop words

[249]:
```
# We are removing stop words

from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

text_data["review_body"] = text_data["review_body"].apply(lambda x: " ".join([i␣
↪for i in x.split() if i not in stop_words]))
```

3.2 perform lemmatization

[250]:
```
#We perform lemmatization on the data

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
text_data["review_body"] = text_data["review_body"].apply(lambda x: " ".
↪join(lemmatizer.lemmatize(i) for i in x.split()))
```

[251]:
```
review_mean_3 = text_data.review_body.apply(lambda x : np.mean(len(str(x)))).
↪mean()
print("\n -x--x- The average length of the reviews in terms of character length␣
↪in the dataset after pre-processing is -x--x- \n", review_mean_3)
print("\n -x--x- The sample reviews after pre-processing -x--x-\n")
print(text_data["review_body"].sample(n=3, random_state=100))
```

```
 -x--x- The average length of the reviews in terms of character length in the
dataset after pre-processing is -x--x-
 189.801875

 -x--x- The sample reviews after pre-processing -x--x-

4471705    ordered coffee maker nov finally delivered mar…
231497     cracked plastic base airtight around lid poor …
1986586      item received scratch used already disappointed
Name: review_body, dtype: object
```

4 TF-IDF Feature Extraction

[252]:
```
# We use train_test_split from sklearn to split the data into 80% training and␣
↪20% testing sets
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

X_train, X_test, y_train, y_test = train_test_split(text_data["review_body"],␣
↪text_data["label"], test_size=0.2, random_state=100)

#We ignore terms that have a document frequency strictly higher 0.7 and␣
↪document frequency strictly lower than 1.
vectorizer = TfidfVectorizer(min_df=1, max_df=0.7)
Xtrain = vectorizer.fit_transform(X_train)
Xtest = vectorizer.transform(X_test)
```

[253]:
```
# we standardize the data using StandardScaler from sklearn
from sklearn.preprocessing import StandardScaler

#Create the instance
sc = StandardScaler(with_mean=False)

#We fit the scaler to the training feauture set only
sc.fit(Xtrain)

#Scale or Transform the training and the testing tests using the scaler that␣
↪was fitted to training data
Xtrain_std = sc.transform(Xtrain)
Xtest_std = sc.transform(Xtest)
```

5 Perceptron

[254]:
```
#implementation of perceptron
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score

# Create a perceptron object with the parameters: 40 iterations (epochs) over␣
↪the data, and a learning rate of 0.1
ppn = Perceptron(max_iter=40, eta0=0.1, random_state=100)

# Fit the model to the standardized data
ppn.fit(Xtrain_std, y_train)
```

```python
# Apply the trained perceptron on the X data to make predicts for the Y test␣
↪data
y_pred_test = ppn.predict(Xtest_std)

#We measure the performance using the "accuracy_score,f1_score,precision_score␣
↪and recall_score"

print("\n -x--x- Accuracy, Precision, Recall, and f1-score on test data␣
↪-x--x-\n")

print(f'Testing Accuracy:{accuracy_score(y_test, y_pred_test):.4f}')
print(f'Testing f1_Score:{f1_score(y_test, y_pred_test):.4f}')
print(f'Testing Precision:{precision_score(y_test, y_pred_test):.4f}')
print(f'Testing recall_score:{recall_score(y_test, y_pred_test):.4f}')

#Apply the trained perceptron on the data to make predicts for the trained data
y_pred_tarin = ppn.predict(Xtrain_std)


print("\n -x--x- Accuracy, Precision, Recall, and f1-score on train data␣
↪-x--x-\n")

print(f'Training Accuracy:{accuracy_score(y_train, y_pred_tarin):.4f}')
print(f'Training f1_Score:{f1_score(y_train, y_pred_tarin):.4f}')
print(f'Training Precision:{precision_score(y_train, y_pred_tarin):.4f}')
print(f'Training recall_score:{recall_score(y_train, y_pred_tarin):.4f}')
```

```
 -x--x- Accuracy, Precision, Recall, and f1-score on test data -x--x-

Testing Accuracy:0.8267
Testing f1_Score:0.8275
Testing Precision:0.8204
Testing recall_score:0.8346

 -x--x- Accuracy, Precision, Recall, and f1-score on train data -x--x-

Training Accuracy:0.9264
Training f1_Score:0.9268
Training Precision:0.9234
Training recall_score:0.9302
```

6 SVM

[255]:
```
#implementation of SVM
from sklearn import svm

#Create a Classifier for svm
clf = svm.LinearSVC() # We are using Linear Kernel

#Train the model using the training sets
clf.fit(Xtrain, y_train)

#Apply the trained svm on Xtrain_std data to make predictions for the test data
y_pred_test = clf.predict(Xtest)


print("\n -x--x- Accuracy, Precision, Recall, and f1-score on test data -x--x-␣
↪\n")

print(f'Testing Accuracy:{accuracy_score(y_test, y_pred_test):.4f}')
print(f'Testing f1_Score:{f1_score(y_test, y_pred_test):.4f}')
print(f'Testing Precision:{precision_score(y_test, y_pred_test):.4f}')
print(f'Testing recall_score:{recall_score(y_test, y_pred_test):.4f}')

#Apply the trained perceptron on the data to make predicts for the trained data
y_pred_tarin = clf.predict(Xtrain)

#We measure the performance using the "accuracy_score,f1_Score,Precision_score␣
↪and recall_score"

print("\n -x--x- Accuracy, Precision, Recall, and f1-score on train data -x--x-␣
↪\n")

print(f'Training Accuracy:{accuracy_score(y_train, y_pred_tarin):.4f}')
print(f'Training f1_Score:{f1_score(y_train, y_pred_tarin):.4f}')
print(f'Training Precision:{precision_score(y_train, y_pred_tarin):.4f}')
print(f'Training recall_score:{recall_score(y_train, y_pred_tarin):.4f}')
```

```
 -x--x- Accuracy, Precision, Recall, and f1-score on test data -x--x-

Testing Accuracy:0.8961
Testing f1_Score:0.8958
Testing Precision:0.8943
Testing recall_score:0.8973

 -x--x- Accuracy, Precision, Recall, and f1-score on train data -x--x-

Training Accuracy:0.9338
```

```
Training f1_Score:0.9338
Training Precision:0.9349
Training recall_score:0.9327
```

7 Logistic Regression

[256]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

# instantiate the model
logreg = LogisticRegression()

# fit the model with data
logreg.fit(Xtrain,y_train)

#Apply the trained Logistic Regression on Xtrain_std data to make predictions␣
↪for the test data
y_pred_test=logreg.predict(Xtest)

#We measure the performance using the "accuracy_score,f1_Score,Precision_score␣
↪and recall_score"
print("\n-x--x- Accuracy, Precision, Recall, and f1-score on test data␣
↪--x--x-\n")

print(f'Testing Accuracy:{accuracy_score(y_test, y_pred_test):.4f}')
print(f'Testing f1_Score:{f1_score(y_test, y_pred_test):.4f}')
print(f'Testing Precision:{precision_score(y_test, y_pred_test):.4f}')
print(f'Testing recall_score:{recall_score(y_test, y_pred_test):.4f}')

#Apply the trained perceptron on the data to make predicts for the trained data
y_pred_tarin = logreg.predict(Xtrain)

#We measure the performance using the "accuracy_score"

print("\n-x--x- Accuracy, Precision, Recall, and f1-score on train data␣
↪-x--x-\n")

print(f'Training Accuracy:{accuracy_score(y_train, y_pred_tarin):.4f}')
print(f'Training f1_Score:{f1_score(y_train, y_pred_tarin):.4f}')
print(f'Training Precision:{precision_score(y_train, y_pred_tarin):.4f}')
print(f'Training recall_score:{recall_score(y_train, y_pred_tarin):.4f}')
```

```
-x--x- Accuracy, Precision, Recall, and f1-score on test data --x--x-

Testing Accuracy:0.9001
Testing f1_Score:0.8993
```

```
Testing Precision:0.9025
Testing recall_score:0.8960

-x--x- Accuracy, Precision, Recall, and f1-score on train data -x--x-

Training Accuracy:0.9143
Training f1_Score:0.9141
Training Precision:0.9171
Training recall_score:0.9111
```

```
/Users/shreenidhihegde/opt/anaconda3/envs/NLP/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:763: 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(
```

8 Naive Bayes

[257]:
```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import MultinomialNB

#Create a Gaussian Classifier
model = MultinomialNB()

# Train the model using the training sets
model.fit(Xtrain, y_train)

#Predict Output
y_pred_test=model.predict(Xtest)

#We measure the performance using the "accuracy_score,f1_Score,Precision_score␣
 ↪and recall_score"
print("\n -x--x- Accuracy, Precision, Recall, and f1-score on test data -x--x-␣
 ↪\n")

print(f'Testing Accuracy:{accuracy_score(y_test, y_pred_test):.4f}')
print(f'Testing f1_Score:{f1_score(y_test, y_pred_test):.4f}')
print(f'Testing Precision:{precision_score(y_test, y_pred_test):.4f}')
print(f'Testing recall_score:{recall_score(y_test, y_pred_test):.4f}')

#Apply the trained perceptron on the data to make predicts for the trained data
```

```
y_pred_tarin = model.predict(Xtrain)

#We measure the performance using the "accuracy_score"

print("\n -x--x- Accuracy, Precision, Recall, and f1-score on train data -x--x-␣
 ↪\n")

print(f'Training Accuracy:{accuracy_score(y_train, y_pred_tarin):.4f}')
print(f'Training f1_Score:{f1_score(y_train, y_pred_tarin):.4f}')
print(f'Training Precision:{precision_score(y_train, y_pred_tarin):.4f}')
print(f'Training recall_score:{recall_score(y_train, y_pred_tarin):.4f}')
```

```
 -x--x- Accuracy, Precision, Recall, and f1-score on test data -x--x-

Testing Accuracy:0.8713
Testing f1_Score:0.8703
Testing Precision:0.8734
Testing recall_score:0.8671

 -x--x- Accuracy, Precision, Recall, and f1-score on train data -x--x-

Training Accuracy:0.8869
Training f1_Score:0.8867
Training Precision:0.8894
Training recall_score:0.8840
```

[ ]:

```

```