CSL7670 : Fundamentals of Machine Learning

Lab Report

INSTRUCTED BY: MR. Anand Mishra

॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

Name: **SHISHEER S KAUSHIK**
Roll Number: **M23IQT006**
Program: **M.Tech in Quantum Technologies**

Chapter 1

Lab-1

1.1 Objective

Objective of this assignment is to brush up on the programming skills required for ML. This assignment contains 16 problems. So accordingly, prepare appropriate inputs or input files as required and submit a detailed Python codes.

1.2 Problem-1

Lists and Loops: Write a Python function that takes a list of integers as input and returns the sum of all even numbers in the list.

Solution 1:

```
def sum_func(n):
    even_sum = []
    for i in n:
        if i%2==0:
            even_sum.append(i)
    return even_sum

sum_list = []

Elements = int(input('Update the required number of Elements: '))
for i in range(Elements):
    iter = int(input())
    sum_list.append(iter)

print('Provided list of numbers were: ', sum_list)

sum_var = sum_func(sum_list)

print('The summation of all even numbers in the provided list is: ',sum(
   sum_var))
```

Output 1:

```
Provided list of numbers were:  [6, 7, 11, 18]
The summation of all even numbers in the provided list is:  24
```

1.3 Problem-2

List Comprehension: Given a list of words, create a new list containing the length of each word using list comprehension.

Solution 2:

```python
# Given list of words
list_of_words = ['IBM-qiskit', 'Google-cirq', 'AWS-bracket', 'Azure-quantum',
    'Qutip']

# Initialize an empty list to store word lengths
length_of_words = []

# Iterate through the words in the list and calculate the length of each word
for word in list_of_words:
    length_of_word = len(word)
    length_of_words.append(length_of_word)

print(length_of_words)
```

Output 2:

```
[10, 11, 11, 13, 5]
```

1.4 Problem-3

NumPy Array Operations: Given two NumPy arrays arr1 = np.array([1,2, 3]) and arr2 = np.array([4, 5, 6]), perform element-wise multiplication and store the result in a new array.

Solution 3:

```python
import numpy as np

# Define arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# element-wise multiplication
array_mul = arr1 * arr2

print(array_mul)
```

Output 3:

```
[44 66 33]
```

1.5 Problem-4

NumPy Array Slicing: Given a NumPy array data = np.arange(1, 21), use array slicing to extract elements from index 5 to index 15.

Solution 4):

```python
import numpy as np

# Create NumPy array
data = np.arange(1, 21)
```

```
# slicing array to extract elements from index 5 to index 15
sliced_data = data[5:16]

print('The Original array data was: {} \nThe Sliced array data is: {}' .
   format(data,sliced_data))
```

Output 4:

```
The Original array data was: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
    17 18 19 20]
The Sliced array data is: [ 6  7  8  9 10 11 12 13 14 15 16]
```

1.6 Problem-5

Data Visualization with Matplotlib: Using Matplotlib, create a line plot to visualize the trend of a stock's closing price over ten days (random data can be used).

Solution 5:

```
import matplotlib.pyplot as plt

# Initiate a random data for the closing prices over ten days
days = [1,2,3,4,5,6,7,8,9,10]
closing_prices = [75,68,80,98,65,75,78,82,86,93]

# Creating a line plot
plt.figure(figsize=(6, 4))  # Set the figure size
plt.plot(days, closing_prices, marker='o', linestyle='-', color='b', label='
   Closing Price')

# labels and title
plt.xlabel('Days')
plt.ylabel('Closing Price')
plt.title('Stock Closing Price Trend Over Ten Days')
plt.xticks(days)
plt.legend()  # legend

plt.show()
```

Figure 1.1: Line plot determining the stock closing price.

1.7 Problem-6

Data Cleaning with Pandas:** Given a Pandas DataFrame with a column containing missing values, write a Python function to replace those missing values with the mean of that column.

Solution 6:

```
import pandas as pd

def fill_missing_with_mean(df, inplace=False):
    # mean for each column
    column_means = df.mean()

    # Fill missing values in each column with the corresponding mean
    if inplace:
        df.fillna(column_means, inplace=True)
        return None
    else:
        df_filled = df.fillna(column_means)
        return df_filled

# sample DataFrame with missing values

data = pd.DataFrame({"A":[12, 4, 5, None, 1],
                     "B":[None, 2, 54, 3, None],
                     "C":[20, 16, None, 3, 8],
                     "D":[14, 3, None, None, 6]})

df = pd.DataFrame(data)

# Call the defined function
filled_df = fill_missing_with_mean(df)

print("\nDataFrame:")
print(df)
print("\nDataFrame filled with means in the missing value:")
print(filled_df)
```

Output 6:

```
DataFrame:
      A     B     C     D
0  12.0   NaN  20.0  14.0
1   4.0   2.0  16.0   3.0
2   5.0  54.0   NaN   NaN
3   NaN   3.0   3.0   NaN
4   1.0   NaN   8.0   6.0

DataFrame filled with means in the missing value:
      A          B      C          D
0  12.0  19.666667  20.00  14.000000
1   4.0   2.000000  16.00   3.000000
2   5.0  54.000000  11.75   7.666667
3   5.5   3.000000   3.00   7.666667
4   1.0  19.666667   8.00   6.000000
```

1.8 Problem-7

Pandas DataFrame Filtering: Using Pandas, filter a DataFrame to only include rows where the 'age' column is greater than 30 and the 'gender' column is 'female'.

Solution 7:

```
import pandas as pd

# DataFrame
data = {
        'person': ['Olivia', 'Bobby', 'Casper', 'Davin', 'Ava', 'Robert', '
           Agusthya', 'Sopia', 'Isabella', 'Martha'],
        'gender': ['female', 'male', 'female', 'male', 'female', 'male', '
           male', 'female', 'female', 'female'],
        'age': [31, 25, 45, 32, 28, 35, 20, 15, 50, 30],
        }

df = pd.DataFrame(data)

# Filter the DataFrame using Query string
filt_df = df.query('age > 30 and gender == "female"')

print('Below is a list of Females above 30 years of age : {}' .format(filt_df
   ))
```

Output 7:

```
     person  gender  age
0    Olivia  female   31
2    Casper  female   45
8  Isabella  female   50
```

1.9 Problem-8

Pandas Grouping: Given a DataFrame with columns 'Country' and 'Population', use Pandas to group the data by 'Country' and calculate the average population for each country.

Solution 8:

```
import pandas as pd

# Sample DataFrame with population data for different centuries
data = {
    'Country': ['Germany', 'Poland', 'Russia'],
    '19th Century': [1200000, 560000, 1005000],
    '20th Century': [25000000, 51000000, 78000000],
    '21st Century': [175000000, 35000000, 80000000]
}

df = pd.DataFrame(data)

# calculate the average for each row
```

```python
def calculate_average(row):
    # Exclude the 'Country' column
    population_values = row[1:]
    return population_values.mean()

# Apply the custom function to each row
df['Average Population'] = df.apply(calculate_average, axis=1)

print(df[['Country', 'Average Population']])
```

Output 8:

```
   Country  Average Population
0  Germany        6.706667e+07
1   Poland        2.885333e+07
2   Russia        5.300167e+07
```

1.10 Problem-9

Data Visualization with Seaborn: Using Seaborn, create a box plot to visualize the distribution of ages for different 'Species' in a DataFrame.

Solution 9:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample dataframe with 'Species' and 'age' columns
data = {'tree_Species': ['Pine', 'Olive', 'Beech', 'Yew', 'Baobab', 'Juniper'
    , 'Redwood'],
        'age': [200, 150, 90, 100, 95, 120, 150]}
df = pd.DataFrame(data)

# Create a box plot
plt.figure(figsize=(9, 5))
sns.boxplot(x='tree_Species', y='age', data=df)

# Set plot title and labels
plt.title('Distribution of Ages for Different Species of Trees')
plt.xlabel('tree_Species')
plt.ylabel('Age')

# Show the plot
plt.show()
```

Figure 1.2: Box Plot showing the different species of tree.

1.11 Problem-10

Data Manipulation with Pandas: Given a Pandas DataFrame containing columns 'Sales' and 'Expenses', create a new column 'Profit' that calculates the profit as 'Sales' minus 'Expenses'.

Solution 10:

```
import pandas as pd

# Sample DataFrame
sample_data = {'Sales': [52050, 75502, 62090, 81280],
        'Expenses': [60000, 48800, 49000, 77700]}
df = pd.DataFrame(sample_data)

# Used assign() method to create a new column 'Profit'
new_df = df.assign(Profit=df['Sales'] - df['Expenses'])

print(new_df)
```

Output 10:

```
   Sales  Expenses  Profit
0  52050     60000   -7950
1  75502     48800   26702
2  62090     49000   13090
3  81280     77700    3580
```

1.12 Problem-11

Dictionary Manipulation: Given a dictionary phonebook containing names as keys and phone numbers as values, write a function to find and return the name(s) of the person(s) with the maximum phone number(s).

Solution 11:

```
def max_phone_numbers_finder(phonebook):
    if not phonebook:
        return None
```

```
    # Find the person with the maximum number of phone numbers
    max_phone_count = 0
    person_with_max_phone_nums = None
    for person, phones in phonebook.items():
        phone_nums = len(phones)
        if phone_nums > max_phone_count:
            max_phone_count = phone_nums
            person_with_max_phone_nums = person

    return person_with_max_phone_nums

# Example phonebook
phonebook = {
    'Buster Keatons': ['8877452478', '8747896512', '9988992510'],
    'Charlie Chaplin': ['7452043210', '1235478625'],
    'Harold LLyold': ['9965485333', '8787854529', '1212121212', '9888777799'
       ],
    'Marilyn Monroe': ['988777779', '9999888877', '1112225557'],
}

person_name = max_phone_numbers_finder(phonebook)
print('The Person with maximum phone numbers in their phonebook is {}' .
   format(person_name))
```

Output 11:

```
The Person with maximum phone numbers in their phonebook is Harold LLyold
```

1.13 Problem-12

Functions and Recursion: Write a recursive Python function to calculate the factorial of a given positive integer.

Solution 12:

```
def Recur_func(n):
   if n == 1:  # Factorial of 1 is 1
       return n
   else:
       return n*Recur_func(n-1)  # Using a factorial number nx(n-1)x...x1

# take input from the user
num = int(input("Enter a number: "))
# check is the number is negative
if num < 0:
   print("Sorry mate! the entered number {} is non-positive number" .format(
      num))
elif num == 0:
   print("The factorial of 0 is 1")
else:
   print("The factorial of {} is {}" .format(num,Recur_func(num)))
```

Output 12:

```
For negative number:
Sorry mate! the entered number -5 is non-positive number

For positive number:
The factorial of 5 is 120
```

1.14 Problem-13

File Handling: Read a text file named "data.txt" that contains one number per line. Write a Python script to calculate the sum of all the numbers and print the result.

Solution 13:

```
# Initialize the tot_sum variable
total_sum = 0

# Read the file to calculate the sum
with open("data.txt", "r") as file:
    for list in file:
        # Convert the list to a number
        number = float(list)
        total_sum += number

# Print the result
print("The sum of all numbers in the file is: {}" .format(total_sum))
```

Output 13:

```
The sum of all numbers in the file is: 5050.0
```

1.15 Problem-14

Object-Oriented Programming: Create a class Rectangle with attributeswidth and height, and methods to calculate the area and perimeter. Test the class by creating objects and performing calculations.

Solution 14:

```
class Rect():
    def __init__(self,width,length,height):
        self.width=width
        self.length=length
        self.height=height

    def area1(self):
        return self.length*self.width
    def peri1(self):
        return 2*(self.length+self.width)
    def vol1(self):
        return self.length*self.width*self.height
```

```
length=int(input('Enter the intended length: '))
width=int(input('Enter the intended width: '))
height=int(input('Enter the intended height: '))
mensuration=Rect(width,length,height)

print('The given value of length is: {}\nThe given value of Width is: {}\nThe
    given value of Height is: {}' .format(length,width,height))
rec_area=mensuration.area1()
print('The Area of Rectangle is: {}' .format(rec_area))
rec_peri=mensuration.peri1()
print('The Perimeter of Rectangle is: {}' .format(rec_peri))
rec_vol=mensuration.vol1()
print('The Volume of Rectangle is: {}' .format(rec_vol))
```

Output 14:

```
Given data:
The given value of length is: 5
The given value of Width is: 5
The given value of Height is: 10

Output data:
The Area of Rectangle is: 25
The Perimeter of Rectangle is: 20
The Volume of Rectangle is: 250
```

1.16 Problem-15

Numpy Array Manipulation: Create a 3×3 identity matrix using NumPy and then convert it into a 1D array.

Solution 15:

```
import numpy as np

arr = np.eye(3)
print('The Identity matrix of 3x3 is : \n')
print(arr)
print('\nThe 1D array of Tdentity Matrix of 3x3 is : \n')
print(arr.reshape(-1))
```

Output 15:

```
The Identity matrix of 3x3 is :

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

The 1D array of Tdentity Matrix of 3x3 is :

[1. 0. 0. 0. 1. 0. 0. 0. 1.]
```

1.17 Problem-16

Data Visualization with Matplotlib: Using Matplotlib, create a bar chart to compare the average scores of students in three different subjects.

Solution 16:

```
import matplotlib.pyplot as plt

# data with scores for each subject w.r.t every student
student_scores = {
    'Rajan': {'Algebra': 80, 'Geometry': 75, 'Calculus': 90},
    'Samar': {'Algebra': 88, 'Geometry': 72, 'Calculus': 78},
    'Doma': {'Algebra': 90, 'Geometry': 90, 'Calculus': 65},
    'Ludwic': {'Algebra': 75, 'Geometry': 78, 'Calculus': 82},
    'Max': {'Algebra': 73, 'Geometry': 85, 'Calculus': 70}
}

# Initialize a dictionary
average_scores = {}

# Calculate average scores for each subject
# Since the structure of the dictionary is consistent for all students, we
   can extract the list of subjects by accessing any student's dictionary.
for subject in student_scores['Doma']:
    total_score = sum(student_scores[student][subject] for student in
       student_scores)
    average_scores[subject] = total_score / len(student_scores)

colors = ['blue', 'green', 'red']

# Plot a bar chart
plt.bar(average_scores.keys(), average_scores.values(), color=colors)

# Adding labels and title
plt.xlabel('Subjects')
plt.ylabel('Average Scores')
plt.title('Average Scores of each student in Different Subjects')
plt.show()
```

Figure 1.3: Histogram showing the avg scores of each student.

Chapter 2

Lab-2

2.1 Objective

The main Objective of this assignment is to get familiarity with K-NN. The first problem is on (Handwritten Digit Classification) Use the code provided for classifying the handwritten digits of the MNIST dataset. Read and understand the code.

2.2 Problem-1(a)

Modify the code so that it uses L1-distance instead of the default L2-distance (Eucleadean).

Solution 1(a):

```
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load MNIST dataset
mnist = fetch_openml('mnist_784')
X = mnist.data
y = mnist.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Define a range of k values to try
k_values = range(1, 7)  # You can adjust the range as needed

best_k = None
best_accuracy = 0.0

for k in k_values:
    # Create a KNN classifier with the current k value and L1-distance (
        Manhattan) or select (p=1)
    knn_classifier = KNeighborsClassifier(n_neighbors=k, metric='manhattan')

    # Train the classifier on the training data
    knn_classifier.fit(X_train, y_train)

    # Predict the labels for the test data
    y_pred = knn_classifier.predict(X_test)

    # Calculate the accuracy of the model
    accuracy = accuracy_score(y_test, y_pred)

    # Check if this k value gives better accuracy
```

```
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

    print(f"Accuracy (k={k}): {accuracy}")

print(f"The best k value is {best_k} with accuracy {best_accuracy}")
```

Output 1(a):

```
Accuracy (k=1): 0.9657142857142857
Accuracy (k=2): 0.9561428571428572
Accuracy (k=3): 0.9661428571428572
Accuracy (k=4): 0.9625
Accuracy (k=5): 0.965
Accuracy (k=6): 0.9611428571428572
The best k value is 3 with accuracy 0.9661428571428572
```

2.3 Problem-2(a)

Display results by showing the image, actual label, and predicted label.

Solution 2(a):

```
# Display some random images and their predicted labels
for idx in range(4):
    random_index = np.random.randint(0, X_test.shape[0])  # Generate a random
        index within the range
    print(random_index)
    print(X_test.shape)
    image = X_test.iloc[random_index, :].values.reshape(28, 28)
    actual_label = y_test.iloc[random_index]
    predicted_label = y_pred[random_index]

    # Generate the Sample images
    plt.figure(figsize=(4, 4))
    plt.imshow(image, cmap='gray')
    plt.title(f"Actual Label: {actual_label}, Predicted Label: {
       predicted_label}")
    plt.axis('off')
    plt.show()
```

Output 2(a):

Figure 2.1: Image generated for label 5.

Figure 2.2: Image sample generated for label 9.

Figure 2.3: Image sample generated for label 8.

Figure 2.4: Image sample generated for label 1.

2.4 Problem-3(a)

Find out a few samples where the predicted label is incorrect.

Solution 3(a):

```
# Display some random images and their predicted labels
for idx in range(70):
    random_index = np.random.randint(0, X_test.shape[0])  # Generate a random
        index within the range
    #print(random_index)
    #print(X_test.shape)
    image = X_test.iloc[random_index, :].values.reshape(28, 28)
    actual_label = y_test.iloc[random_index]
    predicted_label = y_pred[random_index]

    # Check if the predicted label differs from the actual label
    if actual_label != predicted_label:
        plt.figure(figsize=(4, 4))
        plt.imshow(image, cmap='gray')
```

```
plt.title(f"Actual Label: {actual_label}, Predicted Label: {
    predicted_label}")
plt.axis('off')
plt.show()
```

Output 3(a):

Figure 2.5: Mismatched Image sample of actual label 3 with predicted label 1.

Figure 2.6: Mismatched Image sample of actual label 3 with predicted label 8.

Figure 2.7: Mismatched Image sample of actual label 9 with predicted label 7.

2.5 Objective

The second problem is on the (Apple vs Orange) classification. You are given a K-NN code for Apple vs Orange problem. Please read and understand the code. Now perform the following tasks:

2.6 Problem-1(b)

Synthetically increase the dataset size to 50 samples.

Solution 1(b):

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Sample synthetic dataset, 1=Apple, -1=Orange
def gen_rand_data(n):
    rand_samples = n
    samples = []

    # Generating random samples for class 1 (Apple)
    for idx in range(rand_samples // 2):
        redness = np.random.randint(200, 255)
        weight = np.random.randint(100, 140)
        samples.append([redness, weight, 1])

    # Generating random samples for class -1 (Orange)
    for idx in range(rand_samples // 2):
        redness = np.random.randint(10, 50)
        weight = np.random.randint(80, 100)
        samples.append([redness, weight, -1])

    return np.array(samples)

# Generating random 50 sample datasets
data = gen_rand_data(50)
print(data)
```

Output 1(b):

```
[[248 125   1]
 [219 131   1]
 [229 120   1]
 [239 129   1]
 [222 101   1]
 [245 135   1]
 [246 103   1]
 [239 123   1]
 [215 121   1]
 [251 112   1]
 [204 130   1]
 [210 124   1]
 [253 114   1]
 [229 118   1]
 [228 132   1]
 [233 113   1]
 [216 137   1]
 [217 131   1]
 [241 119   1]
 [226 110   1]
 [240 128   1]
 [228 113   1]
 [242 107   1]
 [209 100   1]
 [229 107   1]
...
```

```
 [ 12  91  -1]
 [ 37  91  -1]
 [ 10  80  -1]
 [ 21  80  -1]]
```

```
# List the dimension of the dataset
data.shape
```

```
(50, 3)
```

2.7 Problem-2(b)

Edit the code so that random 80 percentage, 10 percentage, and 10 percentage samples are used for training, testing, and validation respectively.

Solution 2(b):

```
# Separate features (redness and weight) and labels (fruit type)
X = data[:, :2]
y = data[:, 2]

# Split dataset into training (80%), testing (10%), and validation (10%) sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2,
    random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size
    =0.5, random_state=42)
```

Output 2(b):

```
[[ 45  92]
 [239 101]
 [203 122]
 [ 37  92]
 [ 19  82]]
[-1  1  1 -1 -1]
Accuracy: 1.0
```

2.8 Problem-3(b)

Change the value of K to 3, 5, and 7 and compare the validation set and test set results.

Solution 3(b):

```
# Values of k to try
k_values = [3, 5, 7]

for k in k_values:
    # Create a KNN classifier with the current k value
    knn_classifier = KNeighborsClassifier(n_neighbors=k)

    # Train the classifier on the training data
    knn_classifier.fit(X_train, y_train)

    # Predict the fruit types for the validation data
```

```
    y_pred_val = knn_classifier.predict(X_val)

    # Calculate the accuracy of the model on the validation set
    accuracy_val = accuracy_score(y_val, y_pred_val)
    print(f"Validation Accuracy (k={k}):", accuracy_val)

    # Predict the fruit types for the test data
    y_pred_test = knn_classifier.predict(X_test)

    # Calculate the accuracy of the model on the test set
    accuracy_test = accuracy_score(y_test, y_pred_test)
    print(f"Test Accuracy (k={k}):", accuracy_test)
    print()
```

Output 3(b):

```
[[222 136   1]
 [214 129   1]
 [251 103   1]
 [213 117   1]
 [220 101   1]
 [219 101   1]
 [227 108   1]
 [224 108   1]
 [216 124   1]
 [204 117   1]
 [242 130   1]
 [250 106   1]
 [217 104   1]
 [237 101   1]
 [253 106   1]
 [215 104   1]
 [248 137   1]
 [214 125   1]
 [229 129   1]
 [234 103   1]
 [254 106   1]
 [204 106   1]
 [237 115   1]
 [203 103   1]
 [200 123   1]
...

Validation Accuracy (k=7): 1.0
Test Accuracy (k=7): 1.0
```

2.9 Problem-4(b)

Write a code that draws confusion matrices for different K.

Solution 4(b):

```
# Calculate the confusion matrix for validation set
    confmat_val = confusion_matrix(y_val, y_pred_val)
    print("Confusion Matrix (Validation):")
    print(cm_val)
```

```
# Plot the confusion matrix
plt.figure(figsize=(6, 6))
sns.heatmap(confmat_val, annot=True, fmt="d", cmap="Blues")
plt.title(f"Confusion Matrix (Validation) - k={k}")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print()
```

Output 4(b):

```
Validation Accuracy (k=3): 1.0
Confusion Matrix (Validation):
[[4 0]
 [0 1]]
```

Figure 2.8: Confusion Matrix (Validation) for k=3.

```
Validation Accuracy (k=5): 1.0
Confusion Matrix (Validation):
[[4 0]
 [0 1]]
```

Figure 2.9: Confusion Matrix (Validation) for k=5.

```
Validation Accuracy (k=7): 1.0
Confusion Matrix (Validation):
[[4 0]
 [0 1]]
```

Figure 2.10: Confusion Matrix (Validation) for k=7.

Chapter 3

Lab-3

3.1 Objective

The main Objective of this assignment is to get familiarity with perceptron. So, accordingly assignment contains 2 problems on perceptron learning algorithms.

3.2 Problem

You are given a code for perceptron. Please modify it so that, it works for labels given as (+1,-1). It works for the data provided in the attached CSV file. For create a CSV file with three features and two classes manually. Read this CSV file and executive the Perceptron code.

Dataframe imported from .csv file:

```
# Added extra feature as sweetnes: [weight, redness, sweetness
   {[0-100]->{60-90}:apple::{30-50}:orange}]
# The below dataframe are defined in .csv file

220,139,92
236,107,62
226,131,99
210,138,67
36,89,35
45,84,33
40,81,24
49,91,43
41,84,47
```

Solution 1(a):

```
import pandas as pd

# Define the CSV file path
csv_file = "data.csv"

##Read the CSV file into a pandas DataFrame
training_data = pd.read_csv(csv_file)

# Convert the DataFrame to a numpy array
training_data = training_data.values

# Now, 'df' is a DataFrame containing your CSV data
print(training_data)
```

Output 1(a):

```
# Dataframs from .csv are coverted into numpy array

[[220 139  92]
 [236 107  62]
 [226 131  99]
```

```
 [210 138  67]
 [ 36  89  35]
 [ 45  84  33]
 [ 40  81  24]
 [ 49  91  43]
 [ 41  84  47]]
```

Solution 1(b):

```python
import numpy as np

class Perceptron:
    def __init__(self, num_features, learning_rate=0.1, epochs=1000):
        self.weights = np.random.rand(num_features + 1)  # +1 for the bias
           term
        self.learning_rate = learning_rate
        self.epochs = epochs

    def activate(self, x):
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        return self.activate(summation)

    def train(self, training_data, labels):
        for _ in range(self.epochs):
            for inputs, label in zip(training_data, labels):
                prediction = self.predict(inputs)
                update = self.learning_rate * (label - prediction)
                self.weights[1:] += update * inputs
                self.weights[0] += update
                print('updated')

# Labels: 1 for Apple, -1 for Orange
labels = np.array([1, 1, 1, 1,-1,-1,-1,-1,-1])

# Create and train the perceptron
perceptron = Perceptron(num_features=3)
perceptron.train(training_data, labels)

# Test the trained perceptron
test_data = np.array([
    [245,130,90],  # Apple
    [50,30,40],  # Orange
    [222,110,70],   # Apple
    [45,90,30],  # Orange
])

for data in test_data:
    prediction = perceptron.predict(data)
    if prediction == 1:
        print("Apple")
    else:
        print("Orange")
```

Output 1(b):

```
updated
updated
updated
updated
updated
updated
updated
updated
updated
updated
updated
...
Apple
Orange
Apple
Orange
```

Solution 1(c):

```
import numpy as np
import matplotlib.pyplot as plt

# Create and train the perceptron
perceptron = Perceptron(num_features=3)
perceptron.train(training_data, labels)

# Generate points for plotting the decision boundary
x_vals = np.linspace(0, 300, 100)
y_vals = -(perceptron.weights[1] * x_vals + perceptron.weights[0]) /
    perceptron.weights[2]

# Scatter plot of training data
plt.scatter(training_data[labels == 1][:, 0], training_data[labels == 1][:,
    1], color='red', label='Apple')
plt.scatter(training_data[labels == -1][:, 0], training_data[labels == -1][:,
     1], color='orange', label='Orange')

# Plot the decision boundary
plt.plot(x_vals, y_vals, label='Decision Boundary')

plt.xlabel('Weight')
plt.ylabel('Redness')
plt.title('Perceptron Decision Boundary')
plt.legend()
plt.show()
```

Output 1(c):

```
updated
updated
updated
updated
updated
updated
updated
updated
```

```
updated
updated
...
updated
updated
updated
updated
```

Figure 3.1: Perceptron Decision Boundary.

Chapter 4

Lab-4

4.1 Objective

The main Objective of this assignment is to get is to gain familiarity with neural networks (multi-layer perception).

4.2 Problem 1(a)

(Simple MLP) Please go through this blog on developing your first neural network: [Blog] and understand the code.
(A) Draw the model architecture by showing each perceptron, input/output/hidden layers.

Solution 1(a):

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim

# load the dataset, split into input (X) and output (y) variables
dataset = np.loadtxt('pima-indians-diabetes.csv', delimiter=',')
X = dataset[:,0:8]
y = dataset[:,8]


X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32).reshape(-1, 1)

# define the model
class PimaClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.hidden1 = nn.Linear(8, 12)
        self.act1 = nn.ReLU()
        self.hidden2 = nn.Linear(12, 8)
        self.act2 = nn.ReLU()
        self.output = nn.Linear(8, 1)
        self.act_output = nn.Sigmoid()

    def forward(self, x):
        x = self.act1(self.hidden1(x))
        x = self.act2(self.hidden2(x))
        x = self.act_output(self.output(x))
        return x

model = PimaClassifier()
print(model)

# train the model
```

```python
loss_fn   = nn.BCELoss()  # binary cross entropy
optimizer = optim.Adam(model.parameters(), lr=0.001)

n_epochs = 100
batch_size = 10

for epoch in range(n_epochs):
    for i in range(0, len(X), batch_size):
        Xbatch = X[i:i+batch_size]
        y_pred = model(Xbatch)
        ybatch = y[i:i+batch_size]
        loss = loss_fn(y_pred, ybatch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# compute accuracy
y_pred = model(X)
accuracy = (y_pred.round() == y).float().mean()
print(f"Accuracy {accuracy}")

# make class predictions with the model
predictions = (model(X) > 0.5).int()
for i in range(5):
    print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))
```

Output 1(a):

```
PimaClassifier(
  (hidden1): Linear(in_features=8, out_features=12, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=12, out_features=8, bias=True)
  (act2): ReLU()
  (output): Linear(in_features=8, out_features=1, bias=True)
  (act_output): Sigmoid()
)
Accuracy 0.7591145634651184
[6.0, 148.0, 72.0, 35.0, 0.0, 33.599998474121094, 0.6269999742507935, 50.0]
   => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.600000381469727, 0.35100001096725464, 31.0]
   => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.299999237060547, 0.671999990940094, 32.0] =>
   1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.100000381469727, 0.16699999570846558, 21.0]
   => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.099998474121094, 2.2880001068115234, 33.0]
    => 1 (expected 1
```

Figure 4.1: Model Architecture of Neural Network for 'PimaClassifier'.

4.3 Problem 1(b)

(B) Change the code to use only 60 percent, 70 percent, and 80 percent data as training, and report test-set performance for all these three training data set sizes.

Solution 1(b):

```
# Train and evaluate the model for different training data sizes
for size in [0.6, 0.7, 0.8]:
    # Split dataset into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
       random_state=42)

    # Use a portion of the training data as specified by train_size
    train_index = int(len(X_train) * size)
    X_train = X_train[:train_index]
    y_train = y_train[:train_index]

    X_train = torch.tensor(X_train, dtype=torch.float32)
    y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1, 1)

    model = PimaClassifier()

    loss_fn = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    n_epochs = 100
    batch_size = 10

    for epoch in range(n_epochs):
        for i in range(0, len(X_train), batch_size):
            Xbatch = X_train[i:i + batch_size]
            y_pred = model(Xbatch)
            ybatch = y_train[i:i + batch_size]
            loss = loss_fn(y_pred, ybatch)
            optimizer.zero_grad()
```

```
            loss.backward()
            optimizer.step()

    # Compute accuracy on the test set
    X_test = torch.tensor(X_test, dtype=torch.float32)
    y_test = torch.tensor(y_test, dtype=torch.float32).reshape(-1, 1)
    y_pred_test = model(X_test)
    accuracy = (y_pred_test.round() == y_test).float().mean()

    print(f"Training Size: {size * 100}%")
    print(f"Accuracy: {accuracy.item()}")
    print()
```

Output 1(b):

```
Training Size: 60.0%
Accuracy: 0.7337662577629089

Training Size: 70.0%
Accuracy: 0.7402597665786743

Training Size: 80.0%
Accuracy: 0.7272727489471436
```

4.4 Problem 1(c)

(C) Compare BCE loss with MSE loss.

Solution 1(c):

```
# Train and evaluate the model with BCE loss
train_sizes = [0.6, 0.7, 0.8]
print("Using BCE Loss:")
for size in train_sizes:
    loss_fn = nn.BCELoss()  # Binary Cross Entropy Loss
    accuracy = train_and_evaluate(X, y, loss_fn, size)
    print(f"Training Size: {size * 100}%")
    print(f"Accuracy: {accuracy}")
    print()

# Train and evaluate the model with MSE loss
print("Using MSE Loss:")
for size in train_sizes:
    loss_fn = nn.MSELoss()  # Mean Squared Error Loss
    accuracy = train_and_evaluate(X, y, loss_fn, size)
    print(f"Training Size: {size * 100}%")
    print(f"Accuracy: {accuracy}")
    print()
```

Output 1(c):

```
Using BCE Loss:
Training Size: 60.0%
Accuracy: 0.7207792401313782

Training Size: 70.0%
```

```
Accuracy: 0.7272727489471436

Training Size: 80.0%
Accuracy: 0.6558441519737244

Using MSE Loss:
Training Size: 60.0%
Accuracy: 0.7272727489471436

Training Size: 70.0%
Accuracy: 0.7077922224998474

Training Size: 80.0%
Accuracy: 0.7207792401313782
```

4.5 Problem 1(d)

(D) Change the number of hidden layers from 2 to 4 and compare the performance.

Solution 1(d):

```python
# Define the model
class PimaClassifier(nn.Module):
    def __init__(self, tot_hidden_layers):
        super().__init__()
        self.num_hidden_layers = tot_hidden_layers

        # Input layer
        self.layers = [nn.Linear(8, 12)]
        self.layers.append(nn.ReLU())

        # Hidden layers
        for _ in range(tot_hidden_layers - 1):
            self.layers.append(nn.Linear(12, 12))
            self.layers.append(nn.ReLU())

        # Output layer
        self.layers.append(nn.Linear(12, 1))
        self.layers.append(nn.Sigmoid())

        # Create the model using Sequential
        self.model = nn.Sequential(*self.layers)

    def forward(self, x):
        return self.model(x)

# Train and evaluate the model for different training data sizes with 4
   hidden layers
train_sizes = [0.6, 0.7, 0.8]
tot_hidden_layers = 4  # Change the number of hidden layers here
print("Using 4 Hidden Layers:")

for size in train_sizes:
    # Split dataset into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Use a portion of the training data as specified by train_size
train_index = int(len(X_train) * size)
X_train = X_train[:train_index]
y_train = y_train[:train_index]

X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1, 1)

model = PimaClassifier(tot_hidden_layers)

loss_fn = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
n_epochs = 100
batch_size = 10

for epoch in range(n_epochs):
    for i in range(0, len(X_train), batch_size):
        Xbatch = X_train[i:i + batch_size]
        y_pred = model(Xbatch)
        ybatch = y_train[i:i + batch_size]
        loss = loss_fn(y_pred, ybatch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# Compute accuracy on the test set
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32).reshape(-1, 1)
y_pred_test = model(X_test)
accuracy = (y_pred_test.round() == y_test).float().mean()

print(f"Training Size: {size * 100}%")
print(f"Accuracy: {accuracy.item()}")
print()
```

Output 1(d):

```
Using 4 Hidden Layers:
Training Size: 60.0%
Accuracy: 0.6753246784210205

Training Size: 70.0%
Accuracy: 0.7467532753944397

Training Size: 80.0%
Accuracy: 0.7402597665786743
```

4.6 Problem 2(a)

Develop a neural network that works for MNIST handwritten digit classification. (A) Draw the model architecture.

Solution 2(a):

```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Define batch size
batch_size = 64

# Create data loaders for MNIST dataset without custom transformations
train_loader = DataLoader(
    datasets.MNIST('./data', train=True, download=True, transform=transforms.
       ToTensor()),
    batch_size=batch_size, shuffle=True)

test_loader = DataLoader(
    datasets.MNIST('./data', train=False, download=True, transform=transforms
       .ToTensor()),
    batch_size=batch_size, shuffle=False)

# Define the model
class MNISTClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        self.fc1 = nn.Linear(1024, 128)
        self.fc2 = nn.Linear(128, 10)  # 10 classes for digits 0-9

    def forward(self, x):
        x = self.pool(nn.functional.relu(self.conv1(x)))
        x = self.pool(nn.functional.relu(self.conv2(x)))
        x = x.view(-1, 1024)
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = MNISTClassifier()
print(model)

# Train the model
loss_fn = nn.CrossEntropyLoss()  # Cross-entropy loss for multi-class
   classification
optimizer = optim.Adam(model.parameters(), lr=0.001)

n_epochs = 10  # You may need more epochs for better performance

for epoch in range(n_epochs):
    model.train()  # Set the model to training mode
    for batch in train_loader:
        images, labels = batch
        optimizer.zero_grad()
        outputs = model(images)
```

```
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()

# Evaluate the model
model.eval()  # Set the model to evaluation mode
correct = 0
total = 0
with torch.no_grad():
    for batch in test_loader:
        images, labels = batch
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = correct / total
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Output 2(a):

```
MNISTClassifier(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
     =False)
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=1024, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
Accuracy: 99.02%
```

Figure 4.2: Model Architecture of Neural Network for 'MNISTClassifier'.

4.7 Problem 2(b)

(B) Compare its performance with KNN classification.

Solution 2(b):

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Load MNIST dataset for neural network
nn_transform = transforms.Compose([transforms.ToTensor(), transforms.
    Normalize((0.5,), (0.5,))])
nn_train_dataset = datasets.MNIST(root='./data', train=True, transform=
    nn_transform, download=True)
nn_test_dataset = datasets.MNIST(root='./data', train=False, transform=
    nn_transform, download=True)

# Create data loaders for neural network
nn_batch_size = 64
nn_train_loader = DataLoader(nn_train_dataset, batch_size=nn_batch_size,
    shuffle=True)
nn_test_loader = DataLoader(nn_test_dataset, batch_size=nn_batch_size,
    shuffle=False)

# Prepare data for K-Nearest Neighbors
knn_transform = transforms.Compose([transforms.ToTensor()])  # No
    normalization for KNN
knn_train_dataset = datasets.MNIST(root='./data', train=True, transform=
    knn_transform, download=True)
knn_test_dataset = datasets.MNIST(root='./data', train=False, transform=
    knn_transform, download=True)

knn_batch_size = 64
knn_train_loader = DataLoader(knn_train_dataset, batch_size=knn_batch_size,
    shuffle=True)
knn_test_loader = DataLoader(knn_test_dataset, batch_size=knn_batch_size,
    shuffle=False)

# Flatten images for K-Nearest Neighbors
X_train_knn = knn_train_loader.dataset.data.numpy().reshape(-1, 28 * 28)
y_train_knn = knn_train_loader.dataset.targets.numpy()
X_test_knn = knn_test_loader.dataset.data.numpy().reshape(-1, 28 * 28)
y_test_knn = knn_test_loader.dataset.targets.numpy()

# Train and evaluate a K-Nearest Neighbors classifier
k_neighbors = 3  # You can adjust the number of neighbors
knn_classifier = KNeighborsClassifier(n_neighbors=k_neighbors)
knn_classifier.fit(X_train_knn, y_train_knn)
knn_predictions = knn_classifier.predict(X_test_knn)
knn_accuracy = accuracy_score(y_test_knn, knn_predictions)
print(f"K-Nearest Neighbors Accuracy: {knn_accuracy * 100:.2f}%")

# Train and evaluate the CNN model (as previously defined)
cnn_model = MNISTClassifier()
cnn_loss_fn = nn.CrossEntropyLoss()
cnn_optimizer = optim.Adam(cnn_model.parameters(), lr=0.001)
```

```python
cnn_epochs = 10

for epoch in range(cnn_epochs):
    cnn_model.train()
    for batch in nn_train_loader:
        images, labels = batch
        cnn_optimizer.zero_grad()
        cnn_outputs = cnn_model(images)
        cnn_loss = cnn_loss_fn(cnn_outputs, labels)
        cnn_loss.backward()
        cnn_optimizer.step()

cnn_model.eval()
cnn_correct = 0
cnn_total = 0

with torch.no_grad():
    for batch in nn_test_loader:
        images, labels = batch
        cnn_outputs = cnn_model(images)
        _, cnn_predicted = torch.max(cnn_outputs.data, 1)
        cnn_total += labels.size(0)
        cnn_correct += (cnn_predicted == labels).sum().item()

cnn_accuracy = cnn_correct / cnn_total
print(f"CNN Accuracy: {cnn_accuracy * 100:.2f}%")
```

Output 2(b):

```
K-Nearest Neighbors Accuracy: 97.05%

CNN Accuracy: 99.25%
```

Chapter 5

Lab-5

5.1 Objective

The main Objective of this assignment is to gain familiarity with convolutional neural networks.

5.2 Problem 1(a)

(Simple CNN) Explain the following Pytorch Functions:

5.3 Problem 1.1(a)

(i) conv2D

Solution 1.1(a):

```
Conv2D or `nn.Conv2d:` This is a 2D convolutional layer in PyTorch. It is
   used to apply convolutional operations to the input data, which is
   typically an image or a feature map. The constructor takes parameters like
    the number of input channels, the number of output channels (filters),
   and the size of the convolutional kernel.is a dunction in PyTorch that
   performs a 2D convolution operation on a given input tensor. In deep
   learning, convolution neural network (CNN) are widely used for image
   recognition and processing tasks. The convoled function is a fundamental
   building block of CNN that preforms a convolution operation on the input
   tensor to create a feature map.

In simple terms, the Conv2D function tasks an input tensor, a kernel or
   filter tensor and performs a convolution operation to generate a feature
   map. To create a Conv2D layer in PyTorch, we need to specify the input and
    output channels, kernel's size, stride, padding and dilation.
```

5.4 Problem 1.2(a)

(ii) MaxPool2d

Solution 1.2(a):

```
MaxPool2D or `nn.MaxPool2d:` This is a 2D max-pooling layer in PyTorch. It is
    used for downsampling the spatial dimensions of feature maps. It reduces
   the size of the input by selecting the maximum value from a window of a
   specified size as it slides over the input.
```

5.5 Problem 1.3(a)

(iii) Linear

Solution 1.3(a):

```
`nn.Linear:` This is a fully connected (linear) layer in PyTorch. It
   represents the traditional neural network layer where every neuron in the
   current layer is connected to every neuron in the subsequent layer. The
   constructor specifies the input and output sizes (number of neurons in
   each layer).

For instance, it applies a Linear transformation to the incoming data.

Y= x*WT+b

Parameters:
    * in_features_size of each input sample (i.e size of X)

    * out_features_size of each output sample (i.e size of Y)

    * bias if set to false, the layer will not learn an additive bias.
       Default: True
```

5.6 Problem 1.4(a)

(iv) ReLu

Solution 1.4(a):

```
`F.relu:` This is the Rectified Linear Unit (ReLU) activation function from
   the torch.nn.functional module. It is applied element-wise to the output
   of a layer, introducing non-linearity. It replaces all negative values
   with zero and leaves positive values unchanged.To apply a rectified linear
    unit (ReLu) function element-wise on an input tensor, we use torch.nn.
   ReLu. It replaces all the negative elements in the input tensor with 0(
   Zero), and all the non-negative elements are left unchanged.

It supports only real-valued input tensors. ReLu is used as an activation
   function in neural networks.
```

5.7 Problem 1(b)

(C) Edit the code to modify the CNN architecture in the following four steps (call it myCNN)

(i) Instead of 6 activation maps in conv1, use 5 activation maps
(ii) instead of maxpool use average pool
(iii) Instead of 16 activation maps in conv1, use 10 activation maps (iv) Remove fc2 and change fc1 so that it projects to 100 dimensions instead of 120 currently. Rerun the experiment and compare CNN (original code) and myCNN (this code).
(v) Plot the loss function.

Solution 1.1(b):

```
import torch
```

```
import torchvision
import torchvision.transforms as transforms
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Solution 1.2(b):

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
```

Output 1.2(b):

Figure 5.1: GroundTruth: cat plane ship ship

Solution 1.3(b):

```
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```

Output 1.3(b):

```
cat   plane ship  ship
```

Solution 1.4(b):

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 5, 5)
        self.pool = torch.nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(5, 10, 5)
        self.fc1 = nn.Linear(10 * 5 * 5, 100)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        return x


net = Net()
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
loss_values = []
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        loss_values.append(loss.item())
        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}
                ')
            running_loss = 0.0
```

```
plt.figure(figsize=(10, 5))
plt.plot(range(len(loss_values)), loss_values, label='Training Loss')
plt.title('Training Loss Over Time')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output 1.4(b):

```
[1,  2000] loss: 2.830
[1,  4000] loss: 1.956
[1,  6000] loss: 1.684
[1,  8000] loss: 1.610
[1, 10000] loss: 1.594
[1, 12000] loss: 1.546
[2,  2000] loss: 1.475
[2,  4000] loss: 1.478
[2,  6000] loss: 1.475
[2,  8000] loss: 1.447
[2, 10000] loss: 1.457
[2, 12000] loss: 1.404
```

Figure 5.2: Training Loss Graph

Solution 1.5(b):

```
print('Finished Training')
```

Output 1.5(b):

```
Finished Training
```

Solution 1.5(b):

```
dataiter = iter(testloader)
images, labels = next(dataiter)

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4))
    )
```

Figure 5.3: GroundTruth: cat ship ship plane

Solution 1.6(b):

```
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our
   outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct //
   total} %')
```

Output 1.6(b):

```
Accuracy of the network on the 10000 test images: 49 %
```

Solution 1.7(b):

```
# prepare to count predictions for each class
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

Output 1.7(b):

```
Accuracy for class: plane is 48.3 %
Accuracy for class: car   is 73.9 %
Accuracy for class: bird  is 42.9 %
Accuracy for class: cat   is 13.2 %
Accuracy for class: deer  is 27.2 %
Accuracy for class: dog   is 50.8 %
Accuracy for class: frog  is 67.9 %
Accuracy for class: horse is 54.0 %
Accuracy for class: ship  is 71.0 %
Accuracy for class: truck is 49.9 %
```

Chapter 6

Lab-6

6.1 Objective

The main Objective of this assignment is to gain familiarity with convolutional neural networks.

6.2 Problem 2(a)

Understand how to use pre-trained CNN for extracting features and fine-tuning using the following video tutorials and associated codes. TAs will share train/test/-validation images for two categories. Modify the code so that it works for the shared dataset. Report Test Accuracy.

Solution 2(a):

```
import torch, torchvision
from torchvision import datasets, models, transforms
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import time
from torchsummary import summary

import numpy as np
import matplotlib.pyplot as plt
import os

from PIL import Image
# Applying Transforms to the Data
image_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),
        transforms.RandomRotation(degrees=15),
        transforms.RandomHorizontalFlip(),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
    'valid': transforms.Compose([
        transforms.Resize(size=256),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(size=256),
        transforms.CenterCrop(size=224),
```

```
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ])
}


# Load the Data

# Set train and valid directory paths

dataset = 'cnndata'

train_directory = os.path.join(dataset, 'train')
valid_directory = os.path.join(dataset, 'valid')

# Batch size
bs = 32

# Number of classes
num_classes = len(os.listdir(valid_directory))  #10#2#257
print(num_classes)

# Load Data from folders
data = {
    'train': datasets.ImageFolder(root=train_directory, transform=
       image_transforms['train']),
    'valid': datasets.ImageFolder(root=valid_directory, transform=
       image_transforms['valid'])
}

# Get a mapping of the indices to the class names, in order to see the output
    classes of the test images.
idx_to_class = {v: k for k, v in data['train'].class_to_idx.items()}
print(idx_to_class)

# Size of Data, to be used for calculating Average Loss and Accuracy
train_data_size = len(data['train'])
valid_data_size = len(data['valid'])

# Create iterators for the Data loaded using DataLoader module
train_data_loader = DataLoader(data['train'], batch_size=bs, shuffle=True)
valid_data_loader = DataLoader(data['valid'], batch_size=bs, shuffle=True)
```

Output 2(a):

```
2
{0: 'bird', 1: 'horse'}
```

Solution 2(b):

```
train_data_size, valid_data_size
```

Output 2(b):

```
(3360, 220)
```

Solution 2(c):

```python
alexnet = models.alexnet(pretrained=True)

# Freeze model parameters
for param in alexnet.parameters():
    param.requires_grad = False

# Change the final layer of AlexNet Model for Transfer Learning
alexnet.classifier[6] = nn.Linear(4096, num_classes)
alexnet.classifier.add_module("7", nn.LogSoftmax(dim = 1))
alexnet

summary(alexnet, (3, 224, 224))
```

Output 2(c):

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 55, 55]          23,296
              ReLU-2           [-1, 64, 55, 55]               0
         MaxPool2d-3           [-1, 64, 27, 27]               0
            Conv2d-4          [-1, 192, 27, 27]         307,392
              ReLU-5          [-1, 192, 27, 27]               0
         MaxPool2d-6          [-1, 192, 13, 13]               0
            Conv2d-7          [-1, 384, 13, 13]         663,936
              ReLU-8          [-1, 384, 13, 13]               0
            Conv2d-9          [-1, 256, 13, 13]         884,992
             ReLU-10          [-1, 256, 13, 13]               0
           Conv2d-11          [-1, 256, 13, 13]         590,080
             ReLU-12          [-1, 256, 13, 13]               0
        MaxPool2d-13            [-1, 256, 6, 6]               0
AdaptiveAvgPool2d-14            [-1, 256, 6, 6]               0
          Dropout-15                 [-1, 9216]               0
           Linear-16                 [-1, 4096]      37,752,832
             ReLU-17                 [-1, 4096]               0
          Dropout-18                 [-1, 4096]               0
           Linear-19                 [-1, 4096]      16,781,312
             ReLU-20                 [-1, 4096]               0
           Linear-21                    [-1, 2]           8,194
       LogSoftmax-22                    [-1, 2]               0
================================================================
Total params: 57,012,034
Trainable params: 8,194
Non-trainable params: 57,003,840
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 8.37
Params size (MB): 217.48
Estimated Total Size (MB): 226.43
----------------------------------------------------------------
```

Solution 2(d):

```python
# Define Optimizer and Loss Function
loss_func = nn.NLLLoss()
optimizer = optim.Adam(alexnet.parameters())
optimizer
```

Output 2(d):

```
Adam (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    capturable: False
    differentiable: False
    eps: 1e-08
    foreach: None
    fused: None
    lr: 0.001
    maximize: False
    weight_decay: 0
)
```

Solution 2(e):

```
def train_and_validate(model, loss_criterion, optimizer, epochs=25):
    '''
    Function to train and validate
    Parameters
        :param model: Model to train and validate
        :param loss_criterion: Loss Criterion to minimize
        :param optimizer: Optimizer for computing gradients
        :param epochs: Number of epochs (default=25)

    Returns
        model: Trained Model with best validation accuracy
        history: (dict object): Having training loss, accuracy and validation
            loss, accuracy
    '''

    start = time.time()
    history = []
    best_acc = 0.0

    for epoch in range(epochs):
        epoch_start = time.time()
        print("Epoch: {}/{}".format(epoch+1, epochs))

        # Set to training mode
        model.train()

        # Loss and Accuracy within the epoch
        train_loss = 0.0
        train_acc = 0.0

        valid_loss = 0.0
        valid_acc = 0.0

        for i, (inputs, labels) in enumerate(train_data_loader):

            inputs = inputs.to(device)
            labels = labels.to(device)
```

```
        # Clean existing gradients
        optimizer.zero_grad()

        # Forward pass - compute outputs on input data using the model
        outputs = model(inputs)

        # Compute loss
        loss = loss_criterion(outputs, labels)

        # Backpropagate the gradients
        loss.backward()

        # Update the parameters
        optimizer.step()

        # Compute the total loss for the batch and add it to train_loss
        train_loss += loss.item() * inputs.size(0)

        # Compute the accuracy
        ret, predictions = torch.max(outputs.data, 1)
        correct_counts = predictions.eq(labels.data.view_as(predictions))

        # Convert correct_counts to float and then compute the mean
        acc = torch.mean(correct_counts.type(torch.FloatTensor))

        # Compute total accuracy in the whole batch and add to train_acc
        train_acc += acc.item() * inputs.size(0)

        #print("Batch number: {:03d}, Training: Loss: {:.4f}, Accuracy:
           {:.4f}".format(i, loss.item(), acc.item()))


    # Validation - No gradient tracking needed
    with torch.no_grad():

        # Set to evaluation mode
        model.eval()

        # Validation loop
        for j, (inputs, labels) in enumerate(valid_data_loader):
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass - compute outputs on input data using the
               model
            outputs = model(inputs)

            # Compute loss
            loss = loss_criterion(outputs, labels)

            # Compute the total loss for the batch and add it to
               valid_loss
            valid_loss += loss.item() * inputs.size(0)
```

```
                # Calculate validation accuracy
                ret, predictions = torch.max(outputs.data, 1)
                correct_counts = predictions.eq(labels.data.view_as(
                   predictions))

                # Convert correct_counts to float and then compute the mean
                acc = torch.mean(correct_counts.type(torch.FloatTensor))

                # Compute total accuracy in the whole batch and add to
                   valid_acc
                valid_acc += acc.item() * inputs.size(0)

                #print("Validation Batch number: {:03d}, Validation: Loss:
                   {:.4f}, Accuracy: {:.4f}".format(j, loss.item(), acc.item
                   ()))

        # Find average training loss and training accuracy
        avg_train_loss = train_loss/train_data_size
        avg_train_acc = train_acc/train_data_size

        # Find average training loss and training accuracy
        avg_valid_loss = valid_loss/valid_data_size
        avg_valid_acc = valid_acc/valid_data_size

        history.append([avg_train_loss, avg_valid_loss, avg_train_acc,
           avg_valid_acc])

        epoch_end = time.time()

        print("Epoch : {:03d}, Training: Loss: {:.4f}, Accuracy: {:.4f}%, \n\
           t\tValidation : Loss : {:.4f}, Accuracy: {:.4f}%, Time: {:.4f}s".
           format(epoch+1, avg_train_loss, avg_train_acc*100, avg_valid_loss,
            avg_valid_acc*100, epoch_end-epoch_start))

        # Save if the model has best accuracy till now
        #torch.save(model, dataset+'_model_'+str(epoch)+'.pt')

    return model, history

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

num_epochs = 10
trained_model, history = train_and_validate(alexnet, loss_func, optimizer,
   num_epochs)

torch.save(history, dataset+'_history.pt')
```

Output 2(e):

```
Epoch: 1/10
Epoch : 001, Training: Loss: 0.3303, Accuracy: 84.4048%,
        Validation : Loss : 0.1990, Accuracy: 90.4545%, Time: 25.4889s
Epoch: 2/10
Epoch : 002, Training: Loss: 0.2568, Accuracy: 88.8095%,
        Validation : Loss : 0.1280, Accuracy: 94.5455%, Time: 25.3050s
Epoch: 3/10
```

```
Epoch : 003, Training: Loss: 0.2398, Accuracy: 89.8214%,
        Validation : Loss : 0.0954, Accuracy: 96.3636%, Time: 25.2910s
Epoch: 4/10
Epoch : 004, Training: Loss: 0.2307, Accuracy: 90.3869%,
        Validation : Loss : 0.0841, Accuracy: 96.8182%, Time: 25.2939s
Epoch: 5/10
Epoch : 005, Training: Loss: 0.2321, Accuracy: 90.4464%,
        Validation : Loss : 0.1674, Accuracy: 93.1818%, Time: 25.2664s
Epoch: 6/10
Epoch : 006, Training: Loss: 0.2263, Accuracy: 90.5655%,
        Validation : Loss : 0.1968, Accuracy: 91.3636%, Time: 25.3801s
Epoch: 7/10
Epoch : 007, Training: Loss: 0.2199, Accuracy: 90.6845%,
        Validation : Loss : 0.1705, Accuracy: 93.6364%, Time: 25.3098s
Epoch: 8/10
Epoch : 008, Training: Loss: 0.2039, Accuracy: 91.6667%,
        Validation : Loss : 0.1113, Accuracy: 95.9091%, Time: 25.4997s
Epoch: 9/10
Epoch : 009, Training: Loss: 0.2054, Accuracy: 91.0119%,
        Validation : Loss : 0.1657, Accuracy: 94.5455%, Time: 25.2094s
Epoch: 10/10
Epoch : 010, Training: Loss: 0.2323, Accuracy: 90.7143%,
        Validation : Loss : 0.1630, Accuracy: 95.0000%, Time: 25.2258s
```

Solution 2(f):

```
history = np.array(history)
plt.plot(history[:,0:2])
plt.legend(['Tr Loss', 'Val Loss'])
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.ylim(0,1)
plt.savefig(dataset+'_loss_curve.png')
plt.show()
```

Output 2(f):

Figure 6.1: Loss Graph

Solution 2(g):

```
plt.plot(history[:,2:4])
plt.legend(['Tr Accuracy', 'Val Accuracy'])
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.ylim(0,1)
plt.savefig(dataset+'_accuracy_curve.png')
plt.show()
```

Output 2(g):

Figure 6.2: Loss Graph

Solution 2(h):

```
def predict(model, test_image_name):
    '''
    Function to predict the class of a single test image
    Parameters
        :param model: Model to test
        :param test_image_name: Test image

    '''

    transform = image_transforms['test']

    test_image = Image.open(test_image_name)
    plt.imshow(test_image)

    test_image_tensor = transform(test_image)

    if torch.cuda.is_available():
        test_image_tensor = test_image_tensor.view(1, 3, 224, 224).cuda()
    else:
        test_image_tensor = test_image_tensor.view(1, 3, 224, 224)

    with torch.no_grad():
        model.eval()
```

```
# Model outputs log probabilities
out = model(test_image_tensor)
ps = torch.exp(out)
topk, topclass = ps.topk(2, dim=1)
for i in range(2):
    print("Predcition", i+1, ":", idx_to_class[topclass.numpy()[0][i
       ]],", Score: ", topk.numpy()[0][i])
```

Solution 2(i):

```
predict(trained_model,'cnndata/test/bird/30288.png')
```

Output 2(i):

Figure 6.3: Correct Prediction Prediction 1 : bird , Score: 0.97405934, Prediction 2 : horse , Score: 0.025940629

Solution 2(j):

```
predict(trained_model, 'cnndata/test/bird/31708.png')
```

Output 2(j):

Figure 6.4: Correct Prediction Prediction 1 : bird , Score: 0.99908304, Prediction 2 : horse , Score: 0.000916904

Solution 2(k):

```
predict(trained_model, 'cnndata/test/bird/30906.png')
```

Output 2(k):

Figure 6.5: Correct Prediction Prediction 1 : bird , Score: 0.99399734, Prediction 2 : horse , Score: 0.006002598

Solution 2(l):

```
predict(trained_model, 'cnndata/test/horse/32486.png')
```

Output 2(l):

Figure 6.6: Correct Prediction Prediction 1 : bird , Score: 0.99999548, Prediction 2 : horse , Score: 4.512432e-05

Solution 2(m):

```
predict(trained_model, 'cnndata/test/horse/33383.png')
```

Output 2(m):

Figure 6.7: Correct Prediction Prediction 1 : bird , Score: 0.99693775, Prediction 2 : horse , Score: 0.0030622445

Solution 2(n):

```
predict(trained_model, 'cnndata/test/frog/frog.jpeg')
```

Output 2(n):

Figure 6.8: Wrong Prediction Prediction 1 : bird , Score: 1.0, Prediction 2 : horse , Score: 2.740749e-13

Chapter 7

Lab-7

7.1 Objective

The main Objective of this assignment is to gain both theory and practice related to probability theory.

7.2 Problem 1

You are given marks for the Maths for Computing course 2022-23 of ***IITJodhpur***.

7.3 Problem 1(a)

$(\boldsymbol{A})$ Compute the Histogram of this distribution, use bin size = 5

Solution 1(a):

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
from scipy.optimize import curve_fit
import pandas as pd  # Import pandas

# Load data from the 'Marks' column of the Excel file into a NumPy array of
   floats
data = pd.read_excel('marks.xlsx').values.flatten().astype(float)

# Create a histogram
hist, bins, _ = plt.hist(data, bins=5, density=True, alpha=0.6, color='r',
   label='Histogram')

# Add labels and legend
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.legend()
```

Output 1(a):

Figure 7.1: Probability Distributions Graph

7.4 Problem 1(b)

(***B***) Normalize the histogram to obtain probability distribution

Solution 1(b):

```
# Calculate the total count of data points
total_count = len(data)

# Calculate the bin widths
bin_widths = np.diff(bins)

# Normalize the histogram by dividing bin counts by total count and bin width
normalized_hist = hist / (total_count * bin_widths)

# Plot the normalized histogram
plt.bar(bins[:-1], normalized_hist, width=bin_widths, alpha=0.6, color='y',
    label='Normalized Histogram')

# Add labels and legend
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.legend()
```

Output 1(b):

Figure 7.2: Normalized Histogram Graph

7.5 Problem 1(c)

(C) Using the code provided to fit the Gaussian distribution. Write your observations. Is there any issue you see with curve fit?

Solution 1(c):

```
# Define the Gaussian function
def gaussian(x, mu, sigma, A):
    return A * np.exp(-(x - mu)**2 / (2 * sigma**2))

# Fit the data to the Gaussian distribution using curve_fit
mean_estimate = np.mean(data)
stddev_estimate = np.std(data)
# making the data zero mean and std=1
data = (data - mean_estimate)/stddev_estimate

#Normalize marks

amplitude_estimate = 1  # Initial estimate for the amplitude

# Fit the data to the Gaussian distribution using curve_fit
params, covariance = curve_fit(gaussian, np.arange(len(data)), data, p0=[
   mean_estimate, stddev_estimate, amplitude_estimate],maxfev=10000000)

# Extract the parameters of the fitted Gaussian
mu, sigma, A = params

# Generate the x values for the plot
x = np.linspace(0, 100, 1000)

# Calculate the fitted Gaussian curve
fit = gaussian(x, mu, sigma, A)

plt.plot(x, fit, 'r-', label='Fitted Gaussian')

# Add labels and legend
plt.xlabel('Marks')
plt.ylabel('Probability Density')
plt.legend()

# Show the plot
plt.show()

# Print the estimated parameters of the Gaussian distribution
print("Estimated Mean:", mu)
print("Estimated Standard Deviation:", sigma)
```

Output 1(c):

```
Estimated Mean: 54.581397025333175
Estimated Standard Deviation: 10.887011422971973
```

Figure 7.3: Curve fit for Gaussian distribution

$Basedonmyanalysis, IhavegotanEstimatedMeanof\mathbf{54.582}forthedistributionofstudentsco$

7.6 Problem 1(d)

$(\boldsymbol{D})$ Compute the probability of a student getting more than 75/100.

Solution 1(d):

```
# Calculate the probability of getting more than 75/100 marks
probab_75 = 1 - norm.cdf(75, loc=mu, scale=sigma)
print("Probability of students getting more than 75/100 marks:", probab_75)
```

Output 1(d):

```
Probability of students getting more than 75/100 marks: 0.03036190236485703
```

7.7 Problem 2

Here N (a, b) means normal distribution with mean = a and standard deviation = b. Similarly, U[a,b] means uniform distribution between a and b.

7.8 Problem 2(a)

Write a Python code to plot the following distributions: $(\boldsymbol{A})\boldsymbol{N}(\mathbf{0},\mathbf{1})$

Solution 2(a):

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import pandas as pd

# Set random seed for reproducibility
np.random.seed(0)

range_A = {'A': (-4.5, 4.5)}

# Load data from Excel file and convert to a NumPy array
```

```
data = pd.read_excel('marks.xlsx').values.flatten().astype(float)

# Generate data for distribution A (normal distribution)
dist_A = np.random.normal(0, 1, len(data))

# Create a DataFrame for distribution A
df_A = pd.DataFrame({'A': dist_A})

plt.figure(figsize=(12, 8))

# Plot the histogram for distribution A
plt.subplot(2, 2, 1)
plt.hist(df_A['A'], bins=50, density=True, alpha=0.7, color='b')
plt.title('Distribution (A)')
plt.xlabel('Value')
plt.ylabel('Probability Density')

mu, sigma = 0, 1

x = np.linspace(range_A['A'][0], range_A['A'][1], 1000)
pdf = norm.pdf(x, mu, sigma)
plt.plot(x, pdf, 'r-', lw=2, label='PDF (A)')

plt.legend()

plt.tight_layout()
plt.show()
```

Output 2(a):

Figure 7.4: Histogram of Normal distribution for N (0, 1)

7.9 Problem 2(b)

Write a Python code to plot the following distributions: $(\boldsymbol{B})\boldsymbol{U}[\mathbf{0},\mathbf{5}]$

Solution 2(b):

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import pandas as pd
```

```
# Set random seed for reproducibility
np.random.seed(0)

range_B = {'B': (0,5)}

# Load data from Excel file and convert to a NumPy array
data = pd.read_excel('marks.xlsx').values.flatten().astype(float)

# Generate data for distribution A (normal distribution)
dist_B = np.random.uniform(0, 5, len(data))

# Create a DataFrame for distribution A
df_A = pd.DataFrame({'B': dist_B})

plt.figure(figsize=(12, 8))

# Plot the histogram for distribution A
plt.subplot(2, 2, 1)
plt.hist(df_A['B'], bins=50, density=True, alpha=0.7, color='b')
plt.title('Distribution (B)')
plt.xlabel('Value')
plt.ylabel('Probability Density')

plt.tight_layout()
plt.show()
```

Output 2(b):

Figure 7.5: Histogram of Uniform distribution for U [0, 5]

7.10 Problem 2(c)

Write a Python code to plot the following distributions: $(C)N(0, 0.8)$

Solution 2(c):

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import pandas as pd
```

```
# Set random seed for reproducibility
np.random.seed(0)

range_C = {'C': (-3, 3)}

# Load data from Excel file and convert to a NumPy array
data = pd.read_excel('marks.xlsx').values.flatten().astype(float)

# Generate data for distribution A (normal distribution)
dist_C = np.random.normal(0, 0.8, len(data))

# Create a DataFrame for distribution A
df_C = pd.DataFrame({'C': dist_C})

plt.figure(figsize=(12, 8))

# Plot the histogram for distribution A
plt.subplot(2, 2, 1)
plt.hist(df_C['C'], bins=50, density=True, alpha=0.7, color='b')
plt.title('Distribution (C)')
plt.xlabel('Value')
plt.ylabel('Probability Density')

mu, sigma = 0, 0.8

x = np.linspace(range_C['C'][0], range_C['C'][1], 1000)
pdf = norm.pdf(x, mu, sigma)
plt.plot(x, pdf, 'r-', lw=2, label='PDF (C)')

plt.legend()

plt.tight_layout()
plt.show()
```

Output 2(c):

Figure 7.6: Histogram of Normal distribution for N (0, 0.8)

7.11 Problem 2(d)

Write a Python code to plot the following distributions: $\boldsymbol{(D)N(0, 0.6)}$

Solution 2(d):

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import pandas as pd

# Set random seed for reproducibility
np.random.seed(0)

range_D = {'D': (-2.5, 2.5)}

# Load data from Excel file and convert to a NumPy array
data = pd.read_excel('marks.xlsx').values.flatten().astype(float)

# Generate data for distribution A (normal distribution)
dist_D = np.random.normal(0, 0.6, len(data))

# Create a DataFrame for distribution A
df_D = pd.DataFrame({'D': dist_D})

plt.figure(figsize=(12, 8))

# Plot the histogram for distribution A
plt.subplot(2, 2, 1)
plt.hist(df_D['D'], bins=50, density=True, alpha=0.7, color='b')
plt.title('Distribution (D)')
plt.xlabel('Value')
plt.ylabel('Probability Density')

mu, sigma = 0, 0.6

x = np.linspace(range_D['D'][0], range_D['D'][1], 1000)
pdf = norm.pdf(x, mu, sigma)
plt.plot(x, pdf, 'r-', lw=2, label='PDF (D)')

plt.legend()

plt.tight_layout()
plt.show()
```

Output 2(d):

Figure 7.7: Histogram of Normal distribution for N (0, 0.6)

Chapter 8

Lab-8

8.1 Objective

The main Objective of this assignment is to gain familiarity with both theory and practice related to Bayesian learning and performance metrics in ML.

8.2 Problem 1

Using ***NaiveBayesclassifier*** perform naive Bayes classification on the given dataset 'data.csv'.

8.3 Problem 1(a)

To this end, use 70 percent, 20 percentage and 10 percentage of data for training, validation and testing respectively.

Solution 1(a):

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

# Load and preprocess your data
data = pd.read_csv('data.csv')

# Split data into features (X) and class labels (y)
X = data.drop(columns=['Class Label'])  # Features
y = data['Class Label']  # Class labels

# Split data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.7,
    random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size
    =0.2, random_state=42)
```

8.4 Problem 1(b)

For both the categories compute: Accuracy.

Solution 1(b):

```
# Initialize and train the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Validate the model on the validation set
validation_accuracy = nb_classifier.score(X_val, y_val)
```

```python
# Test the model on the test set
test_accuracy = nb_classifier.score(X_test, y_test)

print(f"Validation Accuracy: {validation_accuracy:.2%}")
print(f"Test Accuracy: {test_accuracy:.2%}")
```

Output 1(b):

```
Validation Accuracy: 91.33%
Test Accuracy: 91.43%
Class Means:
Class 0:
Feature-1    152.597163
Feature-2    491.877601
dtype: float64
Class 1:
Feature-1     46.365889
Feature-2    200.164585
dtype: float64
```

8.5 Problem 1(c)

For both the categories compute: (i) Mean (ii) Covariance.

Solution 1(c):

```python
# Calculate the mean and covariance matrix for each class
unique_classes = y_train.unique()
class_means = {}
covariance_matrices = {}

for class_label in unique_classes:
    class_data = X_train[y_train == class_label]

    # Calculate mean for each feature in the class
    class_mean = class_data.mean()
    class_means[class_label] = class_mean

    # Calculate covariance matrix
    covariance_matrix = class_data.cov()
    covariance_matrices[class_label] = covariance_matrix

print("Class Means:")
for class_label, class_mean in class_means.items():
    print(f"Class {class_label}:\n{class_mean}")

print("\nCovariance Matrices:")
for class_label, covariance_matrix in covariance_matrices.items():
    print(f"Class {class_label}:\n{covariance_matrix}")
```

Output 1(c):

```
Covariance Matrices:
Class 0:
             Feature-1     Feature-2
Feature-1  6973.198710    216.252328
```

```
Feature-2   216.252328  85197.431736
Class 1:
            Feature-1     Feature-2
Feature-1  622.695515    -79.126603
Feature-2  -79.126603  13096.433336
```

Chapter 9

Lab-9

9.1 Objective

The main Objective of this assignment is to gain familiarity with K-Means Clustering.

9.2 Problem 1

Understand K-means clustering from the implementation point of view from the provided resource, and use 'data.csv' to perform $\boldsymbol{k - meansclustering}$.

9.3 Problem 1(a)

Finding of the k-means convergence can also be done using 'Elbow Method'
Solution 1(a):

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

custom_data = pd.read_csv("Data.csv")
custom_data.head()

scaler = StandardScaler()
scaler.fit(custom_data)
scaler_data = scaler.transform(custom_data)

# Elbow method

def search_best_clusters(custom_data, maximum_k):
    cluster_centers = []
    k_values = []

    for k in range(1, maximum_k):

        kmeans_model = KMeans(n_clusters = k)
        kmeans_model.fit(custom_data)

        cluster_centers.append(kmeans_model.inertia_)
        k_values.append(k)

    return cluster_centers, k_values

def generate_elbow_plot(clusters_centers, k_values):
```

```
    figure = plt.subplots(figsize = (12,6))
    plt.plot(k_values, clusters_centers, 'o-', color = 'orange')
    plt.xlabel("Number of clusters (k)")
    plt.ylabel("cluster of inertia")
    plt.title("Elbow Plot of KMeans")
    plt.show()

clusters_centers, k_values = search_best_clusters(scaler_data, 12)
```

Output 1(b):

Figure 9.1: Elbow Plot for Various KMeans

9.4 Problem 1(b)

Now, use 'data.csv' to perform k-means clustering. Try different values of k=2,3,4, and 5 and show clustering using appropriate colors in a scatter plot. Also, show cluster centers.

Solution 1(b):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load a sample dataset (you should replace 'sample_data.csv' with your
   dataset)
data_df = pd.read_csv('DATA.csv')

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_df)

cluster_counts = [2, 3, 4, 5]

# Define custom colors for clusters
custom_colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']

for num_clusters in cluster_counts:
    kmeans_model = KMeans(n_clusters=num_clusters, random_state=0)
    cluster_labels = kmeans_model.fit_predict(scaled_data)
    cluster_centers = kmeans_model.cluster_centers_
```

```python
# Create a unique figure for each cluster count
plt.figure(figsize=(6, 6))

for cluster_id in range(num_clusters):
    plt.scatter(
        scaled_data[cluster_labels == cluster_id, 0],
        scaled_data[cluster_labels == cluster_id, 1],
        label=f'Cluster {cluster_id + 1}',
        color=custom_colors[cluster_id % len(custom_colors)]
    )

plt.scatter(
    cluster_centers[:, 0],
    cluster_centers[:, 1],
    c='orange',
    marker='*',
    s=100,
    label='Cluster Centers'
)

plt.title(f'K-means Clustering (k={num_clusters})')
plt.legend()
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Output 1(b):

Figure 9.2: Clustering plot at k = 2

Figure 9.3: Clustering plot at k = 3

Figure 9.4: Clustering plot at k = 4

Figure 9.5: Clustering plot at k = 5

9.5 Problem 2

You are given a collection of Wikipedia article documents. Identify 5 Key- words in these documents manually and use these keywords to represent documents using Bag of Words. Now using these representations, cluster them into 2 clusters. Share your observations and insights.

The identified 5 common keywords among all the passages are:

india, Captain, cricketer, records, awards

Solution 2:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

# Sample documents
documents = [ "india", "Captain", "cricketer", "records", "awards"]

# Convert documents to TF-IDF features
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)

# Perform KMeans clustering with 2 clusters
num_clusters = 2
kmeans = KMeans(n_clusters=num_clusters)
kmeans.fit(tfidf_matrix)

# Get cluster labels for each document
cluster_labels = kmeans.labels_

# Organize documents into clusters
clusters = {}
```

```python
for i, label in enumerate(cluster_labels):
    if label not in clusters:
        clusters[label] = [documents[i]]
    else:
        clusters[label].append(documents[i])

# Print clusters
for cluster , documents in clusters.items():
    print(f"Cluster{cluster+1}:")
    for document in documents:
        print(f"-{document}")
    print()
```

Output 2:

```
Cluster1:
-india
-Captain
-records
-awards
-prime minister

Cluster2:
-cricketer
```

Chapter 10

Lab-10

10.1 Objective

The main Objective of this assignment is to gain familiarity with PCA.

10.2 Problem 1

Take 100 random images from MNIST. Assume there are of size 28×28, that means they can be represented using 784 dimensions.

10.3 Problem 1(a)

Now using the Sklearn library, project them to top-K Principle Components and Reconstruct the images back.

Solution 1(a):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_openml

# Load the MNIST dataset (or any other suitable dataset)
mnist = fetch_openml('mnist_784')
X = mnist.data.astype('float64') / 255.0  # Normalize pixel values to [0, 1]
y = mnist.target.astype(int)

# Define a range of values for K (number of principal components)
k_values = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Plot original and reconstructed images for different values of K
def plot_gallery(original_images, reconstructed_images, n_row=2, n_col=5,
   title=''):
    plt.figure(figsize=(2.5 * n_col, 2 * n_row))
    plt.suptitle(title, size=16)

    for i in range(n_row * n_col):
        plt.subplot(n_row, 2 * n_col, 2 * i + 1)
        plt.imshow(original_images[i], cmap=plt.cm.gray)
        plt.title(f'Original {y[i]}')
        plt.xticks(())
        plt.yticks(())

        plt.subplot(n_row, 2 * n_col, 2 * i + 2)
        plt.imshow(reconstructed_images[i], cmap=plt.cm.gray)
        plt.title(f'Reconstructed {y[i]}')
        plt.xticks(())
        plt.yticks(())
```

```
    plt.tight_layout()
    plt.show()

# Perform PCA and plot original and reconstructed images for each K
for k in k_values:
    # Perform PCA
    pca = PCA(n_components=k)
    X_pca = pca.fit_transform(X)
    X_reconstructed = pca.inverse_transform(X_pca)

    # Reshape the images for plotting
    original_images = X.values.reshape((-1, 28, 28))
    reconstructed_images = X_reconstructed.reshape((-1, 28, 28))

    # Display original and reconstructed images
    plot_gallery(original_images, reconstructed_images, n_row=2, n_col=5,
       title=f'Original and Reconstructed Images (K={k})')
```

Output 1(a):

Original and Reconstructed Images (K=40)

Original and Reconstructed Images (K=50)

Original and Reconstructed Images (K=60)

Original and Reconstructed Images (K=70)

Original and Reconstructed Images (K=80)

Original and Reconstructed Images (K=90)

10.4 Problem 1(b)

Compute the reconstruction loss and plot this against different values of K.

Solution 1(b):

```
# Compute reconstruction loss for each value of K
reconstruction_losses = []
for k in k_values:
    pca = PCA(n_components=k)
    X_pca = pca.fit_transform(X)
    X_reconstructed = pca.inverse_transform(X_pca)
    loss = np.mean((X - X_reconstructed) ** 2)
    reconstruction_losses.append(loss)

# Plot the reconstruction loss against different values of K
plt.plot(k_values, reconstruction_losses, marker='o')
plt.title('Reconstruction Loss vs. Number of Principal Components (K)')
plt.xlabel('Number of Principal Components (K)')
plt.ylabel('Reconstruction Loss')
plt.show()
```

Output 1(b):

Chapter 11

Lab-11

11.1 Objective

The main Objective of this assignment is to gain familiarity with SVM.

11.2 Problem 1

Learn using SVM from the Sklearn documentation. Now, understand the code provided for 0 vs 1 classification. Compare the results across different kernels. Display the confusion matrix for each of the kernels.

Solution 1:

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,
   ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns  # Import seaborn for customizing colors

# Load the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist['data'], mnist['target'].astype(int)

# Filter to only include samples of digit 0 and 1
X_binary = X[(y == 0) | (y == 1)]
y_binary = y[(y == 0) | (y == 1)]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_binary, y_binary,
   test_size=0.2, random_state=42)

# Train SVM classifiers with different kernels
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for kernel in kernels:
    if kernel == 'poly':
        # Polynomial kernel
        svm_model = SVC(kernel='poly', degree=3, gamma='auto', random_state
           =42)
    else:
        # Linear and RBF kernels
        svm_model = SVC(kernel=kernel, gamma='auto', random_state=42)

    # Train the SVM model
    svm_model.fit(X_train, y_train)

    # Predict using the trained model
    y_pred = svm_model.predict(X_test)
```

```python
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy using {kernel} kernel: {accuracy:.4f}")

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=svm_model.classes_)

# Display confusion matrix with custom colorbar
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
    svm_model.classes_)
disp.plot(colorbar=True)

# Show the plot
plt.show()
```

Output:

Figure 11.1: Accuracy using linear kernel: 0.9997

Figure 11.2: Accuracy using poly kernel: 0.9983

Figure 11.3: Accuracy using rbf kernel: 0.5257

Figure 11.4: Accuracy using sigmoid kernel: 0.5257