# DL – 1

# Problem Statement – Real estate agents want help to predict the house price for regions in the

# USA. He gave you the dataset to work on and you decided to use the Linear Regression Model.

# Create a model that will help him to estimate what the house would sell for.

# URL for a dataset: https://github.com/huzaifsayed/Linear-Regression-Model-for-HousePricePrediction/blob/master/USA\_Housing.csv

import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error import matplotlib.pyplot as plt

url = "https://raw.githubusercontent.com/huzaifsayed/Linear-Regression-Model-for-House-Price- Prediction/master/USA\_Housing.csv"

df = pd.read\_csv(url) print(df.head()) print(df.isnull().sum())

X = df.drop(['Price', 'Address'], axis=1) # Assuming 'Address' is not useful for prediction y = df['Price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) model = LinearRegression()

model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred) print(f'Mean Squared Error: {mse}') plt.scatter(y\_test, y\_pred) plt.xlabel("Actual Prices") plt.ylabel("Predicted Prices") plt.title("Actual Prices vs Predicted Prices") plt.show()

# DL- 2

# Build a Multiclass classifier using the CNN model. Use MNIST or any other suitable dataset.

# a. Perform Data Pre-processing b. Define Model and perform training c. Evaluate Results

# using confusion matrix.

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense from tensorflow.keras.utils import to\_categorical

from sklearn.metrics import confusion\_matrix import seaborn as sns

# Load MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() # Data Pre-processing

X\_train = X\_train.reshape(X\_train.shape[0], 28, 28, 1).astype('float32') / 255

X\_test = X\_test.reshape(X\_test.shape[0], 28, 28, 1).astype('float32') / 255 y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test) # Define Model

model = Sequential([

Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'), MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(128, activation='relu'), Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) # Train the model

model.fit(X\_train, y\_train, batch\_size=128, epochs=5, validation\_split=0.1) # Evaluate Results using confusion matrix

y\_pred = model.predict(X\_test) y\_pred\_classes = np.argmax(y\_pred, axis=1) y\_test\_classes = np.argmax(y\_test, axis=1)

conf\_matrix = confusion\_matrix(y\_test\_classes, y\_pred\_classes) plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.arange(10), yticklabels=np.arange(10))

plt.xlabel('Predicted Label') plt.ylabel('True Label') plt.title('Confusion Matrix') plt.show()

# DL – 3

# Design RNN or its variant including LSTM or GRU a) Select a suitable time series dataset.

# Example – predict sentiments based on product reviews b) Apply for predic

import tensorflow as tf

from tensorflow.keras.datasets import imdb from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

from tensorflow.keras.preprocessing.sequence import pad\_sequences max\_features = 10000

maxlen = 100

batch\_size = 32

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features) x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen) model = Sequential() model.add(Embedding(max\_features, 128))

model.add(LSTM(64, dropout=0.2, recurrent\_dropout=0.2)) model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy']) model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=5, validation\_data=(x\_test, y\_test)) score, acc = model.evaluate(x\_test, y\_test, batch\_size=batch\_size)

print(f'Test score: {score}') print(f'Test accuracy: {acc}')

# DL – 4

# Design and implement a CNN for Image Classification a) Select a suitable image

# classification dataset (medical imaging, agricultural, etc.). b) Optimized with different hyperparameters including learning rate, filter size, no. of layers, optimizers, dropouts, etc.

import numpy as np

from tensorflow.keras.datasets import cifar10 from tensorflow.keras.utils import to\_categorical # Load CIFAR-10 dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data() # Preprocess the data

X\_train = X\_train.astype('float32') / 255 X\_test = X\_test.astype('float32') / 255 y\_train = to\_categorical(y\_train, 10) y\_test = to\_categorical(y\_test, 10)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout from tensorflow.keras.optimizers import Adam

def create\_cnn\_model():

model = Sequential([

Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(32, 32, 3)),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)), Flatten(),

Dense(512, activation='relu'), Dropout(0.5),

Dense(10, activation='softmax')

])

return model

def train\_model(learning\_rate, num\_epochs):

model = create\_cnn\_model()

optimizer = Adam(learning\_rate=learning\_rate)

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy']) history = model.fit(X\_train, y\_train, batch\_size=128, epochs=num\_epochs, validation\_split=0.1) return history, model

def evaluate\_model(model):

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0) print(f'Test Loss: {loss:.4f}')

print(f'Test Accuracy: {accuracy:.4f}') learning\_rate = 0.001

num\_epochs = 10

history, model = train\_model(learning\_rate, num\_epochs) evaluate\_model(model)

# DL - 5

# Design and implement Deep Convolutional GAN to generate images of faces/digits from a

# set of given images

import numpy as np

import matplotlib.pyplot as plt

from keras.datasets import fashion\_mnist from keras.models import Sequential

from keras.layers import Dense, Conv2D, Dropout, Flatten, Reshape, Conv2DTranspose,

BatchNormalization, LeakyReLU

# Load data

(train\_x,), (\_, \_) = fashion\_mnist.load\_data() train\_x = (train\_x / 255.) \* 2 - 1

train\_x = train\_x.reshape(-1, 28, 28, 1)

# Create generator generator = Sequential([

Dense(512, input\_shape=[100]), LeakyReLU(alpha=0.2), BatchNormalization(momentum=0.8), Dense(256),

LeakyReLU(alpha=0.2), BatchNormalization(momentum=0.8), Dense(128),

LeakyReLU(alpha=0.2), BatchNormalization(momentum=0.8), Dense(784),

Reshape([28, 28, 1])

])

# Create discriminator discriminator = Sequential([

Conv2D(64, (5, 5), strides=(2, 2), padding='same', input\_shape=(28, 28, 1)), LeakyReLU(alpha=0.2),

Dropout(0.3),

Conv2D(128, (5, 5), strides=(2, 2), padding='same'), LeakyReLU(alpha=0.2),

Dropout(0.3), Flatten(),

Dense(1, activation='sigmoid')

])

# Create GAN

GAN = Sequential([generator, discriminator]) discriminator.compile(optimizer='adam', loss='binary\_crossentropy') discriminator.trainable = False

GAN.compile(optimizer='adam', loss='binary\_crossentropy')

# Training parameters epochs = 5

batch\_size = 100

noise\_shape = 100

# Training loop

for epoch in range(epochs): print(f"Currently on Epoch {epoch + 1}")

for i in range(train\_x.shape[0] // batch\_size):

noise = np.random.normal(size=[batch\_size, noise\_shape]) gen\_image = generator.predict\_on\_batch(noise) train\_dataset = train\_x[i \* batch\_size:(i + 1) \* batch\_size] train\_label = np.ones(shape=(batch\_size, 1)) discriminator.trainable = True

d\_loss\_real = discriminator.train\_on\_batch(train\_dataset, train\_label) train\_label = np.zeros(shape=(batch\_size, 1))

d\_loss\_fake = discriminator.train\_on\_batch(gen\_image, train\_label) noise = np.random.normal(size=[batch\_size, noise\_shape]) train\_label = np.ones(shape=(batch\_size, 1))

discriminator.trainable = False

d\_g\_loss\_batch = GAN.train\_on\_batch(noise, train\_label)

# plotting generated images at the start and then after every 10 epoch if epoch % 10 == 0:

samples = 10

x\_fake = generator.predict(np.random.normal(loc=0, scale=1, size=(samples, 100))) for k in range(samples):

plt.subplot(2, 5, k + 1)

plt.imshow(x\_fake[k].reshape(28, 28), cmap='gray') plt.xticks([])

plt.yticks([]) plt.tight\_layout()

plt.show() print('Training is complete')

# BI – 2

# Data Visualization from Extraction Transformation and Loading (ETL) Process

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris # To load the Iris dataset # Data Extraction (using sklearn.datasets)

iris = load\_iris() # Load the Iris flower dataset # Data Transformation (Optional)

# The Iris data is pre-processed and ready for use # Data Visualization (Multiple Examples)

# Example 1: Sepal Length vs. Sepal Width colored by Species (Scatter Plot) sepal\_length = iris.data[:, 0]

sepal\_width = iris.data[:, 1]

target\_names = iris.target\_names # Get species names

plt.scatter(sepal\_length, sepal\_width, c=iris.target, cmap='plasma') # Color by target species plt.xlabel('Sepal length (cm)')

plt.ylabel('Sepal width (cm)')

plt.title('Iris Flower Dataset - Sepal Dimensions by Species') plt.legend(target\_names)

plt.show()

# Example 2: Distribution of Petal Lengths across Species (Box Plot) petal\_length = iris.data[:, 2] # Petal length feature

plt.boxplot([petal\_length[iris.target == 0], petal\_length[iris.target == 1], petal\_length[iris.target == 2]], notch=True, vert=False, patch\_artist=True, labels=target\_names) # Separate boxes by species plt.xlabel('Petal Length (cm)')

plt.ylabel('Species')

plt.title('Distribution of Petal Length by Iris Species') plt.show()

# BI – 4

# Data Analysis and Visualization using Advanced Excel.

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

dataset = pd.read\_csv("Social\_Network\_Ads.csv")

X = dataset.iloc[:, [2, 3]].values y = dataset.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0) dataset.info()

dataset

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2) classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score cm = confusion\_matrix(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test, y\_pred) error\_rate = 1 - accuracy # Error rate is 1 - Accuracy precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred) print("Confusion Matrix:") print(cm)

print("Accuracy:", accuracy) print("Error Rate:", error\_rate) print("Precision:", precision) print("Recall:", recall)

import seaborn as sns

def plot\_confusion\_matrix(cm): plt.figure(figsize=(8, 6)) sns.heatmap(cm, annot=True) plt.xlabel('Predicted') plt.ylabel('True') plt.title('Confusion Matrix') plt.show() plot\_confusion\_matrix(cm)

y\_test y\_pred

# BI – 5

# Perform the data classification algorithm using any Classification algorithm

import numpy as np import pandas as pd

import matplotlib.pyplot as plt from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler from sklearn.datasets import load\_iris

# Load the Iris dataset iris = load\_iris()

data = iris.data

# Standardize the data (scaling) scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

# Initialize an empty list to store the values of the within-cluster sum of squares (WCSS) wcss = []

# Determine the number of clusters using the elbow method for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=30) kmeans.fit(data\_scaled)

wcss.append(kmeans.inertia\_) # Plot the elbow graph plt.figure(figsize=(8, 6))

plt.plot(range(1, 11), wcss, marker='o', linestyle='--') plt.title('Elbow Method for Optimal k') plt.xlabel('Number of Clusters')

plt.ylabel('WCSS (Within-Cluster Sum of Squares)') plt.grid()

plt.show()