**Exp:1 Techniques for Data Pre-processing: Mean Removal, Scaling, Normalization**

**Required Libraries:**

pip install pandas

pip install -U scikit-learn

**CODE:**

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

file\_path = 'data.csv'

data = pd.read\_csv(file\_path, header=None, names=['Values'])

print("Original Data:")

print(data)

mean\_values = np.mean(data['Values'])

data['Mean\_Removed\_Values'] = data['Values'] - mean\_values

normalized\_values = (data['Values'] - np.min(data['Values'])) / (np.max(data['Values']) - np.min(data['Values']))

data['Normalized\_Values'] = normalized\_values

scaler = ((data['Values'] - max(data['Values']))/max(data['Values']))

print("\nMean Removed Data:")

print(data[['Mean\_Removed\_Values']])

print("\nNormalized Data:")

print(data[['Normalized\_Values']])

print("\nStandardized Data:")

print(scaler)

2.a. Naïve Bayes Classifier

import pandas as pd

f = pd.DataFrame({'Weather':['Sunny', 'Rainy', 'Sunny', 'Sunny'],

'Wind':['Mild', 'Mild', 'High', 'Mild'],

'Temp':['Moderate', 'Mild', 'Moderate', 'Mild'],

'go':['Yes', 'No', 'Yes', 'Yes']})

print(f.columns)

from sklearn.naive\_bayes import GaussianNB as g

from sklearn.preprocessing import LabelEncoder as le

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

l = le()

for i in f.columns:

f[i] = l.fit\_transform(f[i])

x = f.iloc[:, :3]

y = f.iloc[:, 3]

xtr, xte, ytr, yte = tt(x, y, test\_size=0.3)

gg = g()

gg.fit(xtr, ytr)

y\_pred = gg.predict(xte)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(yte, y\_pred))

2.b. Support vector machine

**Required Libraries:**

pip install matplotlib

**CODE:**

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 = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random\_state = 0)

classifier.fit(X\_train, y\_train)

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

from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test,y\_pred)

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],color = ListedColormap(('pink', 'green'))(i), label = j)

plt.title('SVM (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

2.c.Logistic regression

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from matplotlib.colors import ListedColormap

dataset = pd.read\_csv("diabetes.csv")

x = dataset.iloc[:, [4, 7]].values

y = dataset.iloc[:, 8].values

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

sc\_x = StandardScaler()

xtrain = sc\_x.fit\_transform(xtrain)

xtest = sc\_x.transform(xtest)

print (xtrain[0:10, :])

classifier = LogisticRegression(random\_state = 0)

classifier.fit(xtrain, ytrain)

y\_pred = classifier.predict(xtest)

cm = confusion\_matrix(ytest, y\_pred)

print ("Confusion Matrix : \n", cm)

print ("Accuracy : ", accuracy\_score(ytest, y\_pred))

X\_set, y\_set = xtest, ytest

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1,

stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1,

stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(

np.array([X1.ravel(), X2.ravel()]).T).reshape(

X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)')

plt.xlabel('Age')

plt.ylabel('Diabetes')

plt.legend()

plt.show()

2.d.DECISION TREE

import pandas as pd

import math

import numpy as np

data = pd.read\_csv("3-dataset.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

def \_\_init\_\_(self):

self.children = []

self.value = ""

self.isLeaf = False

self.pred = ""

def entropy(examples):

pos = 0.0

neg = 0.0

for \_, row in examples.iterrows():

if row["answer"] == "yes":

pos += 1

else:

neg += 1

if pos == 0.0 or neg == 0.0:

return 0.0

else:

p = pos / (pos + neg)

n = neg / (pos + neg)

return -(p \* math.log(p, 2) + n \* math.log(n, 2))

def info\_gain(examples, attr):

uniq = np.unique(examples[attr])

gain = entropy(examples)

for u in uniq:

subdata = examples[examples[attr] == u]

sub\_e = entropy(subdata)

gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

return gain

def ID3(examples, attrs):

root = Node()

max\_gain = 0

max\_feat = ""

for feature in attrs:

gain = info\_gain(examples, feature)

if gain > max\_gain:

max\_gain = gain

max\_feat = feature

root.value = max\_feat

uniq = np.unique(examples[max\_feat])

for u in uniq:

subdata = examples[examples[max\_feat] == u]

if entropy(subdata) == 0.0:

newNode = Node()

newNode.isLeaf = True

newNode.value = u

newNode.pred = np.unique(subdata["answer"])

root.children.append(newNode)

else:

dummyNode = Node()

dummyNode.value = u

new\_attrs = attrs.copy()

new\_attrs.remove(max\_feat)

child = ID3(subdata, new\_attrs)

dummyNode.children.append(child)

root.children.append(dummyNode)

return root

def printTree(root: Node, depth=0):

for i in range(depth):

print("\t", end="")

print(root.value, end="")

if root.isLeaf:

print(" -> ", root.pred)

print()

for child in root.children:

printTree(child, depth + 1)

def classify(root: Node, new):

for child in root.children:

if child.value == new[root.value]:

if child.isLeaf:

print ("Predicted Label for new example", new," is:", child.pred)

exit

else:

classify (child.children[0], new)

root = ID3(data, features)

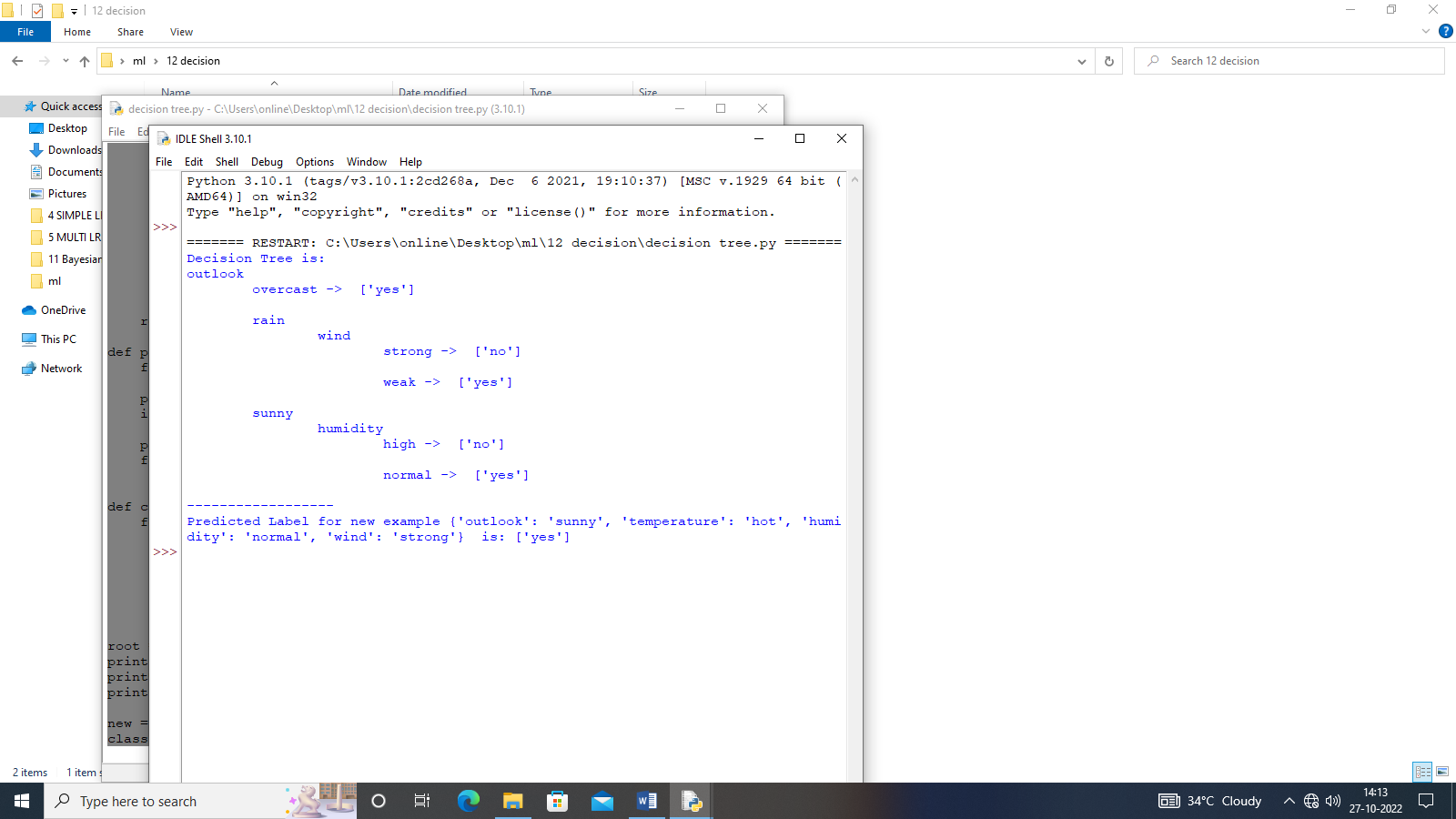
print("Decision Tree is:")

printTree(root)

print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}

classify (root, new)



2.e. Random forest

**Required Libraries:**

pip install seaborn

**CODE:**

import pandas as pd

data=pd.read\_csv("heart.csv")

X =data.iloc[:,[1,2,3,4,5,6,7,8,9,10,11,12]].values

y =data.iloc[:,13].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()

rfc.fit(X\_train, y\_train)

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

from sklearn import metrics

print("Classification Accuracy:", metrics.accuracy\_score(y\_test, y\_pred)\*100)

cm=metrics.confusion\_matrix(y\_test,y\_pred)

print(cm)

import seaborn as sn

from matplotlib import pyplot as plt

plt.figure(figsize=(5,4))

sn.heatmap(cm,annot=True)

plt.xlabel('Predicted value')

plt.ylabel('Actual value')

plt.show()

3.K-MEANS

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import MinMaxScaler

iris = pd.read\_csv("Iris1.csv")

x = iris.iloc[:, [ 1,2,3,4]].values

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

kmeans.fit(x)

wcss.append(kmeans.inertia\_)

kmeans = KMeans(n\_clusters = 3, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

y\_kmeans = kmeans.fit\_predict(x)

plt.scatter(x[y\_kmeans == 0, 0], x[y\_kmeans == 0, 1], s = 100, c = 'blue', label = 'Iris-setosa')

plt.scatter(x[y\_kmeans == 1, 0], x[y\_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')

plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:,1], s = 100, c = 'red', label = 'Centroids')

plt.legend()

plt.show()

**EXPERIMENT 4 NLTK**

**Required Libraries:**

pip install nltk

**CODE:**

import nltk

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.corpus import stopwords

# Download NLTK resources if not already installed

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

# Sample text

text = "The quick brown foxes are jumping over the lazy dogs. The dogs are not amused."

# Tokenization

tokens = word\_tokenize(text)

# Remove stop words

stop\_words = set(stopwords.words('english'))

filtered\_tokens = [word for word in tokens if word.lower() not in stop\_words]

# Stemming

stemmer = PorterStemmer()

stemmed\_tokens = [stemmer.stem(word) for word in filtered\_tokens]

# Lemmatization

lemmatizer = WordNetLemmatizer()

lemmatized\_tokens = [lemmatizer.lemmatize(word) for word in filtered\_tokens]

# Display the results

print("Original Text:", text)

print("\nTokenization:", tokens)

print("\nFiltered Tokens (without stop words):", filtered\_tokens)

print("\nStemmed Tokens:", stemmed\_tokens)

print("\nLemmatized Tokens:", lemmatized\_tokens)

EXPERIMENT – 5(BUILDING BAG OF WORDS MODEL USING NLTK)

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.probability import FreqDist

nltk.download('punkt')

nltk.download('stopwords')

def preprocess\_text(text):

stop\_words = set(stopwords.words('english'))

word\_tokens = word\_tokenize(text)

filtered\_words = [word.lower() for word in word\_tokens if word.isalpha() and word.lower() not in stop\_words]

return filtered\_words

def create\_bow\_model(texts):

all\_words = []

for text in texts:

words = preprocess\_text(text)

all\_words.extend(words)

word\_freq = FreqDist(all\_words)

bow\_model = {word: freq for word, freq in word\_freq.items()}

return bow\_model

# Example usage

texts = [

"The cat sat on the mat, and the mat was comfortable.",

"She sang a sweet song, a song that touched everyone's heart.",

"Coding coding can be challenging, but coding is also incredibly rewarding.",

]

bow\_model = create\_bow\_model(texts)

# Print the Bag of Words model

print("Bag of Words Model:")

for word, freq in bow\_model.items():

print(f"{word}: {freq}")

**Exp 6: Topic Modeling: Identifying Patterns in Text Data**

import csv

import re

def identify\_patterns(csv\_file\_path, column\_name):

patterns = {}

with open(csv\_file\_path, 'r') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

text = row[column\_name]

# Example pattern: finding words that start with 'pattern'

pattern\_matches = re.findall(r'Female', text, flags=re.IGNORECASE)

# Update patterns dictionary with matches

for match in pattern\_matches:

if match in patterns:

patterns[match] += 1

else:

patterns[match] = 1

return patterns

csv\_file\_path = '2b Social\_Network \_Ads.csv' # Update with your CSV file path

column\_name = 'Gender' # Update with the actual column name in your CSV file

result = identify\_patterns(csv\_file\_path, column\_name)

# Display the identified patterns and their counts

for pattern, count in result.items():

print(f"Pattern: {pattern}, Count: {count}")

**Output:**

Pattern: Female, Count: 204

**EXPERIMENT – 7 HIDDEN MARKOV MODEL**

**Required Libraries:**

pip install hmmlearn

**CODE:**import numpy as np

from hmmlearn import hmm

# Step 1: Define Model Parameters

n\_states = 2 # Number of hidden states (Rainy and Sunny)

# Transition matrix (A): Probability of transitioning from one state to another

trans\_matrix = np.array([[0.7, 0.3], [0.4, 0.6]])

# Ensure rows of the transition matrix sum to 1

trans\_matrix /= trans\_matrix.sum(axis=1, keepdims=True)

# Emission matrix (B): Probability of observing an emission given the current state

emission\_matrix = np.array([[0.1, 0.4, 0.5], [0.6, 0.3, 0.1]])

# Initial state probabilities (π): Probability distribution of starting in each state

initial\_probs = np.array([0.6, 0.4])

# Step 2: Create HMM Model

model = hmm.MultinomialHMM(n\_components=n\_states,

startprob\_prior=initial\_probs,

transmat\_prior=trans\_matrix,

n\_iter=100)

# Step 3: Generate Training Data (for simplicity, you can use a pre-existing dataset)

# Observations: 0 - Umbrella, 1 - Jacket, 2 - T-shirt

train\_data = np.array([[0, 1, 2, 0, 1, 2, 0, 2, 1]])

# Reshape the array if needed

train\_data = train\_data.reshape(-1, 1)

# Step 4: Fit the Model

model.fit(train\_data)

# Step 5: Predict States for a New Sequence

new\_data = np.array([[0, 2, 1]]) # Umbrella, T-shirt, Jacket

new\_data = new\_data.reshape(-1, 1)

predicted\_states = model.predict(new\_data)

# Map numerical predictions to weather states

weather\_states = ['Rainy', 'Sunny']

predicted\_states\_text = [weather\_states[state] for state in predicted\_states]

# Display Results

print("Predicted Weather States:", predicted\_states\_text)

Output:   
  
Predicted Weather States: ['Rainy', 'Sunny', 'Rainy']

**EXPERIMENT - 9 A Bot to Play Tic Tac Toe**

def print\_board(board):

for row in board:

print(" | ".join(row))

print("-" \* 5)

def check\_winner(board, player):

# Check rows, columns, and diagonals

for i in range(3):

if all(board[i][j] == player for j in range(3)) or all(board[j][i] == player for j in range(3)):

return True

if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):

return True

return False

def is\_board\_full(board):

return all(board[i][j] != " " for i in range(3) for j in range(3))

def tic\_tac\_toe():

board = [[" " for \_ in range(3)] for \_ in range(3)]

players = ["X", "O"]

current\_player = players[0]

while True:

print\_board(board)

# Get player move

while True:

row = int(input("Enter row (0, 1, or 2): "))

col = int(input("Enter column (0, 1, or 2): "))

if 0 <= row < 3 and 0 <= col < 3 and board[row][col] == " ":

break

else:

print("Invalid move. Try again.")

# Make the move

board[row][col] = current\_player

# Check for a winner

if check\_winner(board, current\_player):

print\_board(board)

print(f"Player {current\_player} wins!")

break

# Check for a tie

if is\_board\_full(board):

print\_board(board)

print("It's a tie!")

break

# Switch to the other player

current\_player = players[1] if current\_player == players[0] else players[0]

if \_\_name\_\_ == "\_\_main\_\_":

tic\_tac\_toe()

**EXPERIMENT-10 SINGLE AND MULTIPLE LAYER PERCEPTRON**

**Required Libraries:**

pip install tensorflow

**CODE:**

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

# Load and preprocess the Iris dataset

X, y = load\_iris(return\_X\_y=True)

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

X\_train, X\_test = StandardScaler().fit\_transform(X\_train), StandardScaler().fit\_transform(X\_test)

# Define and compile a single-layer neural network

model\_single\_layer = models.Sequential([layers.Dense(64, 'relu', input\_shape=(4,)), layers.Dense(3, 'softmax')])

model\_single\_layer.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model\_single\_layer.fit(X\_train, y\_train, epochs=15, validation\_data=(X\_test, y\_test))

# Evaluate the single-layer model

single\_layer\_accuracy = accuracy\_score(y\_test, np.argmax(model\_single\_layer.predict(X\_test), axis=1))

print(f"\nSingle-layer Neural Network - Accuracy: {single\_layer\_accuracy}")

# Define and compile a multi-layer neural network

model\_multi\_layer = models.Sequential([layers.Dense(64, 'relu', input\_shape=(4,)), layers.Dense(32, 'relu'), layers.Dense(3, 'softmax')])

model\_multi\_layer.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model\_multi\_layer.fit(X\_train, y\_train, epochs=15, validation\_data=(X\_test, y\_test))

# Evaluate the multi-layer model

multi\_layer\_accuracy = accuracy\_score(y\_test, np.argmax(model\_multi\_layer.predict(X\_test), axis=1))

print(f"\nMulti-layer Neural Network - Accuracy: {multi\_layer\_accuracy}")

**EXPERIMENT – 11 BUILDING LINEAR REGRESSION USING ANN**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

# Generate some random data for demonstration

np.random.seed(0)

X\_train = np.random.rand(100, 1)

y\_train = 2 \* X\_train + 1 + 0.1 \* np.random.randn(100, 1)

# Build the model

model = tf.keras.Sequential([

tf.keras.layers.Dense(units=1, input\_shape=(1,))

])

# Compile the model

model.compile(optimizer='sgd', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, verbose=0)

# Plot the training loss over epochs

plt.plot(history.history['loss'])

plt.xlabel('Epochs')

plt.ylabel('Mean Squared Error Loss')

plt.title('Training Loss')

plt.show()

# Make predictions on new data

X\_test = np.array([[0.2], [0.5], [0.8]])

predictions = model.predict(X\_test)

# Print the predictions

for i in range(len(X\_test)):

print(f"Input: {X\_test[i][0]}, Predicted Output: {predictions[i][0]}")

**12. IMAGE CLASSIFIER AN APPLICATION OF DEEP LEARNING**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D

# Load and preprocess the MNIST dataset

mnist = tf.keras.datasets.mnist

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0  # Normalize pixel values

# Define the CNN model

model = Sequential([

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

    MaxPooling2D(pool\_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D(pool\_size=(2, 2)),

    Flatten(),

    Dense(64, activation='relu'),

    Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=5, validation\_split=0.1)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(f'\nTest accuracy: {test\_acc}')