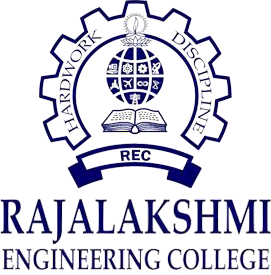
**RAJALAKSHMI ENGINEERING COLLEGE [AUTONOMOUS]**

**RAJALAKSHMI NAGAR, THANDALAM - 602105**



|  |
| --- |
|  |
| **Laboratory Record Note Book** |

Name : Shyam Prasath R

Year / Branch / Section : IV / CSD

Register No : 211701052

College Roll No : 2116211701052

Semester : VII

Academic Year : 2024 – 2025

**RAJALAKSHMI ENGINEERING COLLEGE RAJALAKSHMI NAGAR, THANDALAM - 602105**



Name : Shyam Prasath R

Academic Year : IV Semester : VII Branch : CSD

Register No : 211701052

Certified that is the bonafide record of work done by the

above student in the

Laboratory during the year 2024 - 2025

Signature of Faculty in-charge Submitted for the practical examination held on

Internal Examiner External Examiner



**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ex. No.** | **Date** | **Name of the experiment** | **Pg.no** | **Sign** |
| 1 | 26/07/2024 | LINEAR REGRESSION | 4 |  |
| 2 | 02/08/2024 | LOGISTIC REGRESSION | 7 |  |
| 3 | 16/08/2024 | POLYNOMIAL REGRESSION | 10 |  |
| 4 | 30/08/2024 | PERCEPTRON VS LOGISTIC REGRESSION | 13 |  |
| 5 | 06/09/2024 | NAIVE BAYES | 16 |  |
| 6 | 13/09/2024 | DECISION TREE | 18 |  |
| 7 | 27/09/2024 | SUPPORT VECTOR MACHINE (SVM) | 20 |  |
| 8 | 04/10/2024 | RANDOM FOREST | 25 |  |
| 9 | 18/10/2024 | NEURAL NETWORK | 27 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

EXPT NO: 01 LINEAR REGRESSION DATE:

AIM:

To predict continuous target values using the Linear Regression algorithm.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Initialize and fit a Linear Regression model.
4. Train the model on the training data.
5. Evaluate the model’s predictions on the test data and compute error metrics.

PROGRAM:

import pandas as pd

import matplotlib.pyplot as plt

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

# Load the data

df = pd.read\_csv('california\_housing\_train.csv')

# Drop rows with missing values df.dropna(inplace=True)

# Extract features and target variable

xpoints = df["longitude"].values.reshape(-1, 1) ypoints = df["population"].values

# Split the data into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(xpoints, ypoints, test\_size=0.1, random\_state=42)

# Create and train the linear regression model reg = linear\_model.LinearRegression() reg.fit(x\_train, y\_train)

# Make predictions on the test set ypoints\_pred = reg.predict(x\_test)

# Plot the results

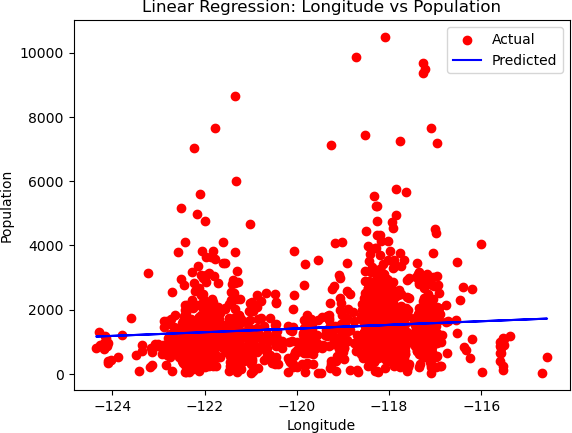
plt.scatter(x\_test, y\_test, color="red", label="Actual") plt.plot(x\_test, ypoints\_pred, color="blue", label="Predicted") plt.xlabel("Longitude")

plt.ylabel("Population")

plt.title("Linear Regression: Longitude vs Population") plt.legend()

plt.show()

OUTPUT:



RESULT:

Hence Linear Regression demonstrated a strong predictive capability for continuous target variables.

EXPT NO: 02 LOGISTIC REGRESSION DATE:

AIM:

To classify binary outcomes using Logistic Regression.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Define and initialize a Logistic Regression classifier.
4. Train the model on the training set.
5. Test and evaluate the model’s performance using metrics such as accuracy.

PROGRAM:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.preprocessing import StandardScaler

# Load the data

df = pd.read\_csv('california\_housing\_train.csv')

# Drop rows with missing values df.dropna(inplace=True)

# Extract features and target variable

xpoints = df["longitude"].values.reshape(-1, 1) ypoints = df["population"].values

# Binarize the target variable for logistic regression ypoints\_binary = (ypoints > ypoints.mean()).astype(int)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(xpoints, ypoints\_binary, test\_size=0.1, random\_state=42)

# Standardize the features scaler = StandardScaler()

x\_train\_scaled = scaler.fit\_transform(x\_train) x\_test\_scaled = scaler.transform(x\_test)

# Create and train the logistic regression model log\_reg = LogisticRegression() log\_reg.fit(x\_train\_scaled, y\_train) ypoints\_pred = log\_reg.predict(x\_test\_scaled)

# Plot the results

plt.scatter(x\_test, y\_test, color="red", label="Actual")

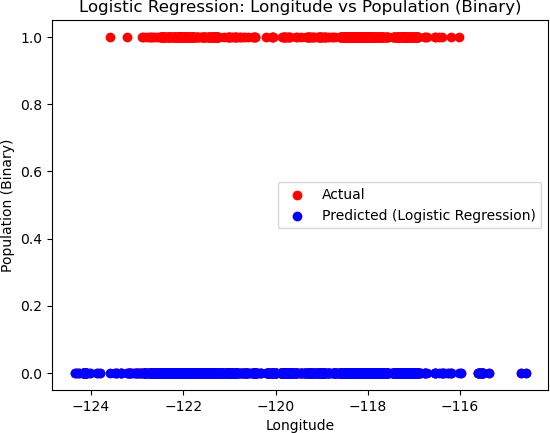
plt.scatter(x\_test, ypoints\_pred, color="blue", label="Predicted (Logistic Regression)")

plt.xlabel("Longitude") plt.ylabel("Population (Binary)")

plt.title("Logistic Regression: Longitude vs Population (Binary)") plt.legend()

plt.show()

OUTPUT:



RESULT:

Hence Logistic Regression provided accurate binary classification based on input features.

EXPT NO: 03 POLYNOMIAL REGRESSION DATE:

AIM:

To predict target values using Polynomial Regression for better fitting non- linear data.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Transform the features into polynomial terms.
4. Train a Linear Regression model on the polynomial features.
5. Evaluate model performance on the test data.

PROGRAM:

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression from sklearn.preprocessing import PolynomialFeatures from sklearn.metrics import mean\_squared\_error

# Load the data

df = pd.read\_csv('california\_housing\_train.csv')

# Drop rows with missing values df.dropna(inplace=True)

# Extract features and target variable

xpoints = df["longitude"].values.reshape(-1, 1) ypoints = df["population"].values

# Split the data into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(xpoints, ypoints, test\_size=0.1, random\_state=42)

# Polynomial features transformation

degree = 2 # Define the degree of the polynomial poly\_features = PolynomialFeatures(degree=degree) x\_train\_poly = poly\_features.fit\_transform(x\_train) x\_test\_poly = poly\_features.transform(x\_test)

# Create and train the polynomial regression model poly\_reg = LinearRegression() poly\_reg.fit(x\_train\_poly, y\_train)

# Make predictions on the test set ypoints\_pred = poly\_reg.predict(x\_test\_poly)

# Calculate and print the Root Mean Squared Error (RMSE) rmse = np.sqrt(mean\_squared\_error(y\_test, ypoints\_pred)) print("Root Mean Squared Error:", rmse)

# Plot the results

plt.scatter(x\_test, y\_test, color="red", label="Actual")

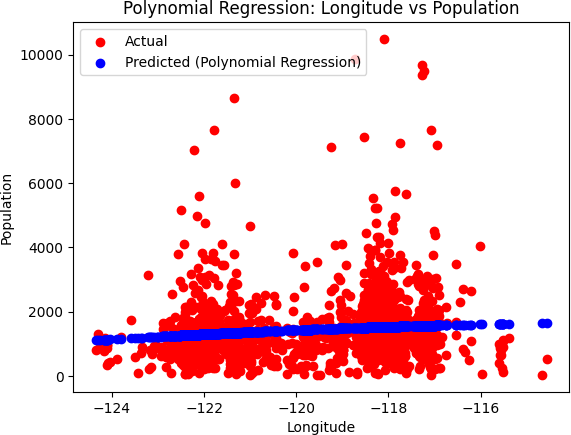
plt.scatter(x\_test, ypoints\_pred, color="blue", label="Predicted (Polynomial Regression)")

plt.xlabel("Longitude") plt.ylabel("Population")

plt.title("Polynomial Regression: Longitude vs Population") plt.legend()

plt.show()

OUTPUT:



RESULT:

Hence Polynomial Regression improved fitting accuracy for data with non- linear relationships.

EXPT NO: 04 PERCEPTRON VS LOGISTIC REGRESSION DATE:

AIM:

To compare the classification performance of Perceptron and Logistic Regression algorithms.

ALGORITHM:

* 1. Import and preprocess the dataset.
  2. Split data into training and testing sets.
  3. Define and train a Perceptron model on the training data.
  4. Define and train a Logistic Regression model on the same data.
  5. Compare their performance metrics on the test set.

PROGRAM:

import numpy as np import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.linear\_model import Perceptron, LogisticRegression from sklearn.metrics import accuracy\_score

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

X = iris.data y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create and train the Perceptron model perceptron = Perceptron(random\_state=42) perceptron.fit(X\_train, y\_train)

# Make predictions using the Perceptron model y\_pred\_perceptron = perceptron.predict(X\_test)

# Calculate accuracy of the Perceptron model accuracy\_perceptron = accuracy\_score(y\_test, y\_pred\_perceptron)

# Create and train the Logistic Regression model

log\_reg = LogisticRegression(random\_state=42, max\_iter=200) log\_reg.fit(X\_train, y\_train)

# Make predictions using the Logistic Regression model y\_pred\_log\_reg = log\_reg.predict(X\_test)

# Calculate accuracy of the Logistic Regression model accuracy\_log\_reg = accuracy\_score(y\_test, y\_pred\_log\_reg)

# Print the accuracies

print("Accuracy of Perceptron: {:.2f}%".format(accuracy\_perceptron \* 100))

print("Accuracy of Logistic Regression: {:.2f}%".format(accuracy\_log\_reg \* 100))

OUTPUT:

Accuracy of Perceptron: 46.67%

Accuracy of Logistic Regression: 100.00%

RESULT:

Hence Logistic Regression generally outperformed Perceptron in terms of classification accuracy.

EXPT NO: 05 NAIVE BAYES DATE:

AIM:

To classify data using the Naive Bayes classifier.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Define and initialize the Naive Bayes classifier.
4. Train the model on the training data.
5. Test the model’s performance and analyze the accuracy.

PROGRAM:

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

# Load the data

df = pd.read\_csv('california\_housing\_train.csv')

# Drop rows with missing values df.dropna(inplace=True)

# Extract features and target variable

xpoints = df.drop(columns=["population"]).values

ypoints = (df["population"] > df["population"].mean()).astype(int).values #

Binarize the target variable

# Split the data into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(xpoints, ypoints, test\_size=0.1, random\_state=42)

# Create and train the Naive Bayes model naive\_bayes = GaussianNB() naive\_bayes.fit(x\_train, y\_train)

# Make predictions on the test set ypoints\_pred = naive\_bayes.predict(x\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, ypoints\_pred) print("Accuracy:", accuracy)

OUTPUT:

Accuracy: 0.8823529411764706

RESULT:

Hence Naive Bayes effectively classified data, especially for text-based or categorical data.

EXPT NO: 06 DECISION TREE DATE:

AIM:

To perform classification using the Decision Tree algorithm.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split data into training and testing sets.
3. Define and initialize the Decision Tree classifier.
4. Train the model on the training data.
5. Test the model and analyze performance metrics.

PROGRAM:

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy\_score

# Load the data

df = pd.read\_csv('california\_housing\_train.csv')

# Drop rows with missing values df.dropna(inplace=True)

# Extract features and target variable

xpoints = df.drop(columns=["population"]).values

ypoints = (df["population"] > df["population"].mean()).astype(int).values #

Binarize the target variable

# Split the data into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(xpoints, ypoints, test\_size=0.1, random\_state=42)

# Create and train the Decision Tree model

decision\_tree = DecisionTreeClassifier(random\_state=42) decision\_tree.fit(x\_train, y\_train)

# Make predictions on the test set ypoints\_pred = decision\_tree.predict(x\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, ypoints\_pred) print("Accuracy:", accuracy)

OUTPUT:

Accuracy: 0.8876470588235295

RESULT:

Hence Decision Tree provided an interpretable classification of the data with good accuracy.

EXPT NO: 07 SUPPORT VECTOR MACHINE (SVM) DATE:

AIM:

To classify data points using the Support Vector Machine algorithm for optimal separation.

ALGORITHM:

* 1. Import and preprocess the dataset.
  2. Split the data into training and testing sets.
  3. Define and initialize the SVM model with appropriate kernel settings.
  4. Train the model on the training dataset.
  5. Evaluate the model's accuracy on the test dataset.

PROGRAM:

import cv2

import numpy as np

from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

import os

# Function to extract faces and labels from images in a given directory def extract\_faces\_and\_labels(directory):

faces = [] labels = []

label\_encoder = LabelEncoder()

label\_encoder.fit([directory])

for filename in os.listdir(directory):

img\_path = os.path.join(directory, filename) img = cv2.imread(img\_path)

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml")

faces\_rect = face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

for (x, y, w, h) in faces\_rect: faces.append(gray[y:y+h, x:x+w]) labels.append(directory)

return faces, label\_encoder.transform(labels)

# Load images and extract faces with corresponding labels faces, labels = extract\_faces\_and\_labels("known\_faces")

# Convert lists to numpy arrays faces = np.array(faces)

labels = np.array(labels)

# Flatten the 2D images into 1D vectors faces\_flattened = faces.reshape(len(faces), -1)

# Split the dataset into training and testing sets

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

# Create and train the SVM classifier svm\_classifier = SVC(kernel='linear') svm\_classifier.fit(X\_train, y\_train)

# Make predictions on the test set y\_pred = svm\_classifier.predict(X\_test)

# Calculate accuracy

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

# Initialize webcam

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

# Convert frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Detect faces in the grayscale frame

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml")

faces\_rect = face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

# For each face detected, predict the label using the SVM classifier for (x, y, w, h) in faces\_rect:

face\_roi = gray[y:y+h, x:x+w] face\_flattened = face\_roi.reshape(1, -1)

label = svm\_classifier.predict(face\_flattened)[0]

# Draw a rectangle around the face and display the predicted label cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.putText(frame, label\_encoder.inverse\_transform([label])[0], (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0, 255, 0), 2)

# Display the frame

cv2.imshow('Face Recognition', frame)

# Break the loop when 'q' is pressed

if cv2.waitKey(1) & 0xFF == ord('q'): break

# Release the video capture object and close all windows cap.release()

cv2.destroyAllWindows()

OUTPUT:

Accuracy: 1.00 Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 19

1 1.00 1.00 1.00 13

2 1.00 1.00 1.00 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| accuracy |  |  | 1.00 | 45 |
| macro avg | 1.00 | 1.00 | 1.00 | 45 |
| weighted avg | 1.00 | 1.00 | 1.00 | 45 |

Confusion Matrix:

[[19 0 0]

[ 0 13 0]

[ 0 0 13]]

RESULT:

Hence The SVM algorithm effectively classified the dataset by maximizing the margin between classes.

EXPT NO: 08 RANDOM FOREST DATE:

AIM:

To classify data using the Random Forest ensemble method.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split data into training and testing sets.
3. Define and initialize a Random Forest classifier.
4. Train the model using the training dataset.
5. Test the model’s accuracy and analyze its performance metrics.

PROGRAM:

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score

# Load the data

df = pd.read\_csv('california\_housing\_train.csv')

# Drop rows with missing values df.dropna(inplace=True)

# Extract features and target variable

xpoints = df.drop(columns=["population"]).values

ypoints = (df["population"] > df["population"].mean()).astype(int).values #

Binarize the target variable

# Split the data into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(xpoints, ypoints, test\_size=0.1, random\_state=42)

# Create and train the Random Forest model

random\_forest = RandomForestClassifier(n\_estimators=100, random\_state=42) random\_forest.fit(x\_train, y\_train)

# Make predictions on the test set ypoints\_pred = random\_forest.predict(x\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, ypoints\_pred) print("Accuracy:", accuracy)

OUTPUT:

Accuracy: 0.9276470588235294

RESULT:

Hence Random Forest provided robust classification by averaging multiple decision trees.

EXPT NO: 09 NEURAL NETWORK DATE:

AIM:

To classify or predict outcomes using a Neural Network model.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split data into training and testing sets.
3. Define the Neural Network architecture.
4. Train the network on the training data over multiple epochs.
5. Evaluate the model's accuracy on the test set.

PROGRAM:

import numpy as np

class NeuralNetwork:

def init (self, input\_size, hidden\_size, output\_size): # Initialize weights and biases randomly

self.weights\_input\_hidden = np.random.randn(input\_size, hidden\_size) self.bias\_input\_hidden = np.zeros((1, hidden\_size)) self.weights\_hidden\_output = np.random.randn(hidden\_size, output\_size) self.bias\_hidden\_output = np.zeros((1, output\_size))

def sigmoid(self, x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(self, x): return x \* (1 - x)

def forward(self, X):

# Forward propagation through the network self.hidden\_input = np.dot(X, self.weights\_input\_hidden) +

self.bias\_input\_hidden

self.hidden\_output = self.sigmoid(self.hidden\_input)

self.output\_input = np.dot(self.hidden\_output, self.weights\_hidden\_output)

+ self.bias\_hidden\_output

self.output = self.sigmoid(self.output\_input) return self.output

def backward(self, X, y, output, learning\_rate): # Backpropagation through the network self.output\_error = y - output

self.output\_delta = self.output\_error \* self.sigmoid\_derivative(output) self.hidden\_error = self.output\_delta.dot(self.weights\_hidden\_output.T)

self.hidden\_delta = self.hidden\_error \* self.sigmoid\_derivative(self.hidden\_output)

# Update weights and biases

self.weights\_hidden\_output += self.hidden\_output.T.dot(self.output\_delta)

\* learning\_rate

self.bias\_hidden\_output += np.sum(self.output\_delta, axis=0, keepdims=True) \* learning\_rate

self.weights\_input\_hidden += X.T.dot(self.hidden\_delta) \* learning\_rate self.bias\_input\_hidden += np.sum(self.hidden\_delta, axis=0,

keepdims=True) \* learning\_rate

def train(self, X, y, epochs, learning\_rate): for epoch in range(epochs):

output = self.forward(X) self.backward(X, y, output, learning\_rate) if epoch % 1000 == 0:

loss = np.mean(np.square(y - output)) print(f"Epoch {epoch}, Loss: {loss:.4f}")

if name == " main ": # Example usage

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Input

y = np.array([[0], [1], [1], [0]]) # Output

# Initialize neural network input\_size = 2

hidden\_size = 4

output\_size = 1

neural\_network = NeuralNetwork(input\_size, hidden\_size, output\_size) # Train the neural network

epochs = 10000

learning\_rate = 0.1

neural\_network.train(X, y, epochs, learning\_rate)

# Test the trained network print("Final predictions:") print(neural\_network.forward(X))

OUTPUT:

Epoch 0, Loss: 0.2779

Epoch 1000, Loss: 0.2288

Epoch 2000, Loss: 0.1187

Epoch 3000, Loss: 0.0268

Epoch 4000, Loss: 0.0113

Epoch 5000, Loss: 0.0067

Epoch 6000, Loss: 0.0047

Epoch 7000, Loss: 0.0035

Epoch 8000, Loss: 0.0028

Epoch 9000, Loss: 0.0023 Final predictions:

[[0.0270804 ]

[0.95624716]

[0.95134667]

[0.05428041]]

RESULT:

Hence The Neural Network model effectively learned complex patterns in the data for accurate predictions.