Practical-3

AIM: Perform the data classification using classification algorithm.

Classification is a supervised learning technique used to categorize data into predefined classes or groups based on input features.

For example:
✔ Spam Detection: Classify emails as Spam or Not Spam
✔ Disease Prediction: Classify patients as Healthy or Diseased
✔ Image Recognition: Identify objects as Dog, Cat, or Human

Program Code:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

# Load the dataset
dataset = load_iris()

# Split the dataset into training (75%) and testing (25%) sets
X_train, X_test, y_train, y_test = train_test_split(dataset["data"], dataset["target"], random_state=0)

# Initialize KNN classifier with k=1
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)  # Train the model

# Print header
print("\n{:^10} {:^20} {:^20}".format("Index", "Actual Class", "Predicted Class"))
print("-" * 50)

# Loop through test samples and predict
for i in range(len(X_test)):
    x = X_test[i]
    x_new = np.array([x])
    prediction = kn.predict(x_new)

    actual_label = dataset["target_names"][y_test[i]]
    predicted_label = dataset["target_names"][prediction[0]]

    print("{:^10} {:^20} {:^20}".format(i, actual_label, predicted_label))

# Print overall accuracy
accuracy = kn.score(X_test, y_test)
print("\nModel Accuracy:", round(accuracy * 100, 2), "%")
```

Practical-4

AIM: Perform the data clustering using a clustering algorithm.

Clustering is an unsupervised machine learning technique used to group similar data points together.

- The algorithm finds patterns in data without predefined labels.
- K-Means clustering is one of the most popular clustering methods.

Program Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

# Load the dataset
iris = load_iris()
X = iris.data  # Only features, no labels

# Apply K-Means Clustering
k = 3  # We know Iris has 3 species
kmeans = KMeans(n_clusters=k, random_state=0)
y_kmeans = kmeans.fit_predict(X)  # Predict cluster labels

# Visualize the clusters using only first two features
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', edgecolors='k', s=100, label="Clusters")
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=300, c='red', marker='X', label="Centroids")

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('K-Means Clustering on Iris Dataset')
plt.legend()
plt.show()

# Print cluster labels
for i in range(len(X)):
    print(f"Data Point {i+1}: Cluster {y_kmeans[i]}")
```

Practical-5

AIM:Perform the Linear regression on given data warehouse data.

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It provides valuable

insights for prediction and data analysis. This article will explore its types, assumptions, implementation, advantages, and evaluation metrics.

Program Code:

```
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.metrics import mean_squared_error, r2_score

# Load the dataset
df = pd.read_csv("advertising_data.csv")

# Extract independent variable (Ad Budget) and dependent variable (Sales)
X = df.iloc[:, 0].values.reshape(-1, 1)  # Feature: Ad Budget
y = df.iloc[:, 1].values  # Target: Sales

# Split dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict sales on test data
y_pred = model.predict(X_test)

# Print the regression equation
print(f"Linear Regression Equation: Y = {model.coef_[0]:.2f}X + {model.intercept_:.2f}")

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
print(f"R² Score: {r2:.4f}")

# Plot actual data vs predicted data
plt.scatter(X, y, color='blue', label="Actual Data")
plt.plot(X, model.predict(X), color='red', linestyle='dashed', label="Regression Line")
plt.xlabel("Advertising Budget ($1000s)")
plt.ylabel("Sales ($1000s)")
plt.title("Linear Regression: Ad Budget vs Sales")
plt.legend()
plt.show()
```

Practical-6

AIM:Perform the Logistic regression on given data warehouse data.

Logistic Regression is a **supervised learning algorithm** used for **classification problems**. Unlike **Linear Regression**, which predicts continuous values, **Logistic Regression predicts categorical values** (such as Yes/No, 0/1, Spam/Not Spam).

Program Code:

```
# Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from math import exp
plt.rcParams["figure.figsize"] = (10, 6)

# Download the dataset
# Source of dataset - https://www.kaggle.com/rakeshrau/social-network-ads
# !wget
"https://drive.google.com/uc?id=15WAD9_4CpUK6EWmgWVXU8YMnyYLKQvW8&export=download"
-O data.csv -q

# Load the data
data = pd.read_csv("/content/sample_data/data1.csv")
data.head()
# Visualizing the dataset
plt.scatter(data['Age'], data['Purchased'])
plt.show()

# Divide the data to training set and test set
X_train, X_test, y_train, y_test = train_test_split(data['Age'], data['Purchased'], test_size=0.20)
# Creating the logistic regression model

# Helper function to normalize data
def normalize(X):
    return X - X.mean()

# Method to make predictions
def predict(X, b0, b1):
    return np.array([1 / (1 + exp(-1*b0 + -1*b1*x)) for x in X])

# Method to train the model
def logistic_regression(X, Y):

    X = normalize(X)

    # Initializing variables
```

```
    b0 = 0
    b1 = 0
    L = 0.001
    epochs = 300

    for epoch in range(epochs):
        y_pred = predict(X, b0, b1)
        D_b0 = -2 * sum((Y - y_pred) * y_pred * (1 - y_pred))  # Derivative of loss wrt b0
        D_b1 = -2 * sum(X * (Y - y_pred) * y_pred * (1 - y_pred))  # Derivative of loss wrt b1
        # Update b0 and b1
        b0 = b0 - L * D_b0
        b1 = b1 - L * D_b1

    return b0, b1
# Training the model
b0, b1 = logistic_regression(X_train, y_train)

# Making predictions
X_test_norm = normalize(X_test)
y_pred = predict(X_test_norm, b0, b1)
y_pred = [1 if p >= 0.5 else 0 for p in y_pred]

plt.clf()
plt.scatter(X_test, y_test)
plt.scatter(X_test, y_pred, c="red")
plt.show()

# The accuracy
accuracy = 0
for i in range(len(y_pred)):
    if y_pred[i] == y_test.iloc[i]:
        accuracy += 1
print(f"Accuracy = {accuracy / len(y_pred)}")
# Making predictions using scikit learn
from sklearn.linear_model import LogisticRegression

# Create an instance and fit the model
lr_model = LogisticRegression()
lr_model.fit(X_train.values.reshape(-1, 1), y_train.values.reshape(-1, 1))

# Making predictions
y_pred_sk = lr_model.predict(X_test.values.reshape(-1, 1))

plt.clf()
plt.scatter(X_test, y_test)
plt.scatter(X_test, y_pred_sk, c="red")
plt.show()

# Accuracy
print(f"Accuracy = {lr_model.score(X_test.values.reshape(-1, 1), y_test.values.reshape(-1, 1))}")
```

Practical-7

AIM:Write a python program to read data from csv file and perform simple data analysis and generate insights.

Program Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Read CSV File
df = pd.read_csv("/content/sample_data/sales_data.csv")
#df.columns = df.columns.str.strip()  # Remove any spaces

# Step 2: Display Basic Information
print("\n First 5 Rows of the Dataset:")
print(df.head())

print("\n Dataset Summary:")
print(df.describe())

# Step 3: Check for Missing Values
print("\n  Missing Values in Dataset:")
print(df.isnull().sum())

# Step 4: Data Insights

# 4.1 Find the product with highest sales
best_selling_product = df.groupby("Product")["Sales"].sum().idxmax()
print(f"\n Best Selling Product: {best_selling_product}")

# 4.2 Average Profit Per Product
avg_profit = df.groupby("Product")["Profit"].mean()
print("\n Average Profit Per Product:")
print(avg_profit)

# 4.3 Correlation Between Ad Budget and Sales
correlation = df["Ad Budget ($1000s)"].corr(df["Sales"])
print(f"\n Correlation between Ad Budget and Sales: {correlation:.2f}")

# Step 5: Data Visualization

# 5.1 Sales Distribution Per Product
plt.figure(figsize=(8, 5))
sns.barplot(x="Product", y="Sales", data=df, palette="viridis")
plt.title("Total Sales Per Product")
plt.xlabel("Product")
plt.ylabel("Total Sales")
plt.show()
```

```
# 5.2 Ad Budget vs. Sales Scatter Plot
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df["Ad Budget ($1000s)"], y=df["Sales"], color="blue")
plt.title("Ad Budget vs Sales")
plt.xlabel("Ad Budget ($1000s)")
plt.ylabel("Sales ($1000s)")
plt.show()

# 5.3 Profit Distribution
plt.figure(figsize=(8, 5))
sns.boxplot(x="Product", y="Profit", data=df, palette="coolwarm")
plt.title("Profit Distribution Per Product")
plt.show()
```