**Week -6**

**40. Market Basket Analysis using Association Rules (Apriori Algorithm)**

**install.packages("arules")**

# Sample dataset: Transactions

transactions <- list(

c("Milk", "Bread"),

c("Milk", "Butter"),

c("Bread", "Butter"),

c("Milk", "Bread", "Butter")

)

# Convert to transaction format

trans <- as(transactions, "transactions")

# Apply Apriori algorithm

rules <- apriori(trans, parameter = list(supp = 0.5, conf = 0.6))

# Display rules

inspect(rules)

**41. Creation of a Data Warehouse**

install.packages("RSQLite")

library(RSQLite)

# Create a database

conn <- dbConnect(SQLite(), "data\_warehouse.db")

# Create a table

dbExecute(conn, "CREATE TABLE IF NOT EXISTS sales (id INTEGER PRIMARY KEY, product TEXT, quantity INTEGER, price REAL)")

# Insert sample data

dbExecute(conn, "INSERT INTO sales (product, quantity, price) VALUES ('Milk', 10, 20.0)")

dbExecute(conn, "INSERT INTO sales (product, quantity, price) VALUES ('Bread', 5, 15.0)")

# Fetch data

data <- dbGetQuery(conn, "SELECT \* FROM sales")

print(data)

# Close connection

dbDisconnect(conn)

**43. Hierarchical Clustering Algorithm**

# Sample dataset

data <- matrix(c(5, 3, 10, 15, 15, 12, 24, 10, 30, 30, 85, 70, 71, 80, 60, 78), ncol=2, byrow=TRUE)

# Apply hierarchical clustering

hc <- hclust(dist(data), method = "ward.D2")

# Plot dendrogram

plot(hc, main="Hierarchical Clustering Dendrogram", xlab="Data Points", sub="")

**44. Bayesian Classification**

# Sample dataset

X <- data.frame(Feature1 = c(1, 2, 3, 4, 5), Feature2 = c(20, 22, 21, 40, 50))

y <- factor(c(0, 0, 0, 1, 1))

# Train Naïve Bayes model

model <- naiveBayes(y ~ ., data = X)

# Predict on a new data point

new\_data <- data.frame(Feature1 = 3, Feature2 = 35)

prediction <- predict(model, new\_data)

print(prediction)

**45. Decision Tree Induction Algorithm**

# Sample dataset

X <- data.frame(Feature1 = c(1, 2, 3, 4, 5), Feature2 = c(10, 20, 30, 40, 50))

y <- factor(c(0, 0, 1, 1, 1))

# Train Decision Tree model

tree\_model <- rpart(y ~ ., data = X, method = "class", control = rpart.control(cp = 0,minsplit = 2))

plot(tree\_model)

text(tree\_model, use.n = TRUE)

**46. Support Vector Machine (SVM)**

# Sample dataset

X <- data.frame(Feature1 = c(1, 2, 3, 4, 5), Feature2 = c(10, 20, 30, 40, 50))

y <- factor(c(0, 0, 1, 1, 1))

# Train SVM model

svm\_model <- svm(y ~ ., data = X, kernel="linear")

# Predict on a new value

new\_data <- data.frame(Feature1 = 3, Feature2 = 35)

prediction <- predict(svm\_model, new\_data)

print(prediction)

**47. Applications of Classification for Web Mining**

install.packages("tm")

install.packages("e1071")

library(tm)

library(e1071)

# Sample dataset

messages <- c("Win a lottery now!", "Get discount on products", "Hello friend, how are you?", "Meeting at 5 PM")

labels <- factor(c(1, 1, 0, 0)) # 1 = Spam, 0 = Not Spam

# Convert text to document-term matrix

corpus <- Corpus(VectorSource(messages))

dtm <- DocumentTermMatrix(corpus)

data <- as.data.frame(as.matrix(dtm))

# Train Naïve Bayes classifier

model <- naiveBayes(data, labels)

# Predict a new message

test\_message <- Corpus(VectorSource("Win free money now!"))

test\_dtm <- as.data.frame(as.matrix(DocumentTermMatrix(test\_message, control=list(dictionary=Terms(dtm)))))

prediction <- predict(model, test\_dtm)

print(prediction)

**Assessment-6**

**Linear SVM Implementation:**

* **Implement a linear SVM classifier using Python and the scikit-learn library. Train the model on a given dataset and evaluate its accuracy.**
* **Dataset: A 2D dataset with two classes (e.g., sklearn’s make blobs).**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Step 1: Generate a 2D dataset with two classes

X, y = make\_blobs(n\_samples=200, centers=2, random\_state=42, cluster\_std=2.0)

# Step 2: Split the 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)

# Step 3: Train the Linear SVM model

svm\_model = SVC(kernel="linear")

svm\_model.fit(X\_train, y\_train)

# Step 4: Make predictions and evaluate accuracy

y\_pred = svm\_model.predict(X\_test)

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

print(f"Model Accuracy: {accuracy:.2f}")

# Step 5: Plot the decision boundary

def plot\_decision\_boundary(model, X, y):

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 100),

np.linspace(y\_min, y\_max, 100))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3)

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors="k")

plt.title("Linear SVM Decision Boundary")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

# Call the function to visualize decision boundary

plot\_decision\_boundary(svm\_model, X, y)