Bahria University,

Karachi Campus

A logo with text on it

Description automatically generated

LAB EXPERIMENT NO.

04

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **01** | Using python implement Decision Tree Algorithm on Diabetes Dataset the chances of diabetes in a person. visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn. |
| **02** | Using KNIME implement Task # 01. |
| 03 | Using python perform the parameter tuning to optimize the Decision Tree performance and compare the results with task # 1. |

Submitted On:

13-03-2024

(Date: DD/MM/YYYY)

**Task No. 01**: Using python implement Decision Tree Algorithm on Diabetes Dataset the chances of diabetes in a person. visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

**Solution:**

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

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

df = pd.read\_csv("/content/diabetes (1).csv")

# Update the specified columns

selected\_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

df = df[selected\_columns]

# Split the dataset into features (X) and target variable (y)

X = df.drop(columns=["Outcome"])

y = df["Outcome"]

# 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)

# Initialize decision tree classifier with Gain Ratio

clf = DecisionTreeClassifier(criterion='entropy', splitter='best')

# Train the classifier on the training data

clf.fit(x\_train, y\_train)

# Make predictions on the testing data

y\_pred = clf.predict(x\_test)

# Evaluate the classifier

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

print("Accuracy:", accuracy)

# Print classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Print confusion matrix

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

# Select the specified columns

selected\_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

df = df[selected\_columns]

# Split the dataset into features (X) and target variable (y)

X = df.drop(columns=["Outcome"])

y = df["Outcome"]

# 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)

# Initialize decision tree classifier with Gain Ratio

clf = DecisionTreeClassifier(criterion='gini', splitter='best')

# Train the classifier on the training data

clf.fit(x\_train, y\_train)

# Make predictions on the testing data

y\_pred = clf.predict(x\_test)

# Evaluate the classifier

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

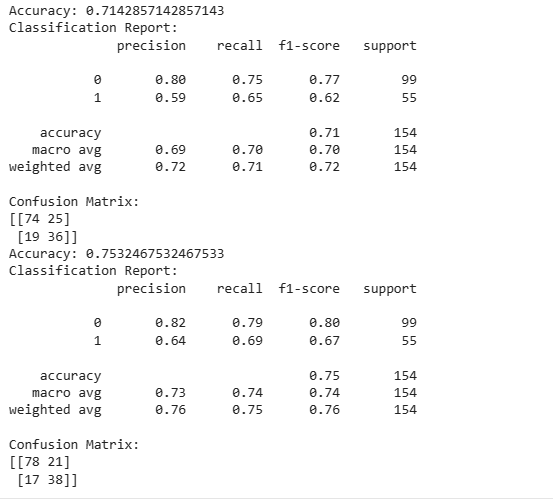
print("Accuracy:", accuracy)

# Print classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Print confusion matrix

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

**Output:**

import matplotlib.pyplot as plt

import seaborn as sns

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

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

A blue squares with white text

Description automatically generatedplt.xlabel("Predicted Label", fontsize=12)

plt.ylabel("True Label", fontsize=12)

plt.title("Confusion Matrix", fontsize=14)

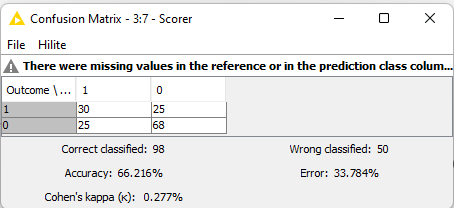
plt.xticks(rotation=45)

plt.show()

**Task No. 02**: Using Knime implement Task # 01.

**A diagram of a process

Description automatically generatedSolution:**

**Output:**

**A screenshot of a computer

Description automatically generated**

**Task No. 03**: Using python to perform the parameter tuning to optimize the Decision Tree performance and compare the results with task # 1.

**Solution:**

from sklearn.model\_selection import GridSearchCV

param\_grid = {

    'max\_depth': [5, 10, 20],

    'min\_samples\_leaf': [1, 2, 4],

    'min\_samples\_split': [2, 5, 10],

}

clf = DecisionTreeClassifier(criterion='gini', splitter='best')

grid\_search = GridSearchCV(clf, param\_grid, cv=5)

grid\_search.fit(x\_train, y\_train)

print("Best parameters:", grid\_search.best\_params\_)

best\_clf = grid\_search.best\_estimator\_

y\_pred = best\_clf.predict(x\_test)

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

print("Accuracy:", accuracy)

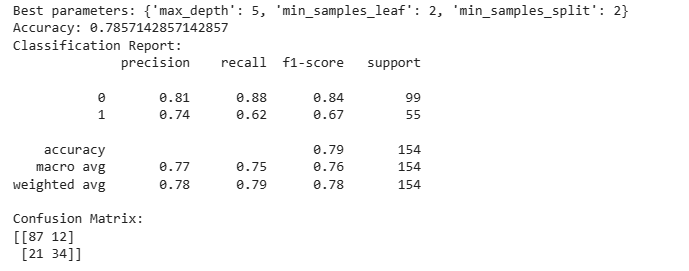
print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

**Output:**

****