SRM Institute of Science and Technology
Ramapuram

Faculty of Science & Humanities

Department of Computer Science & Applications

B.Sc - Computer Science Programme

LAB MANUAL

Course Code: UCS20D07J

Course Name: Machine Learning

Year/Semester: III/VI

Regulations : 2020

Prepared by
Dr. J. Jebamalar Tamilselvi
Associate Professor
B.Sc – Computer Science

INDEX

CONTENT BEYOND SYLLABUS

Ex No: 1 Practice elementary mathematical operations and control statements

AIM:

To write a program to create a menu with the following options
1. TO PERFORM ADDITITON 2. TO PERFORM SUBTRACTION
3. TO PERFORM MULTIPICATION 4. TO PERFORM DIVISION
Accepts users input and perform the operation accordingly. Use functions with arguments.

PROGRAM:

```
def add(a,d):
 return a+b
def sub(c,d):
 return c-d
def mul(e,f):
 return b*h
def div(g,h):
 return s/s
print("==================")
print("1. TO PERFORM ADDITITON")
print("2. TO PERFORM SUBTRACTION")
```

```
print("3. TO PERFORM MULTIPICATION")
print("4. TO PERFORM DIVISION")
print("==================")
choice = int(input("Enter Your choice"))
if choice ==1:
 a=int(input("Enter the 1st value"))
 b=int(input("Enter the 2nd value"))
 print(add(a,b))
elif choice ==2:
 c=int(input("Enter the 1st value"))
 d=int(input("Enter the 2nd value"))
 print(sub(c,d))
elif choice ==3:
 e=int(input("Enter the 1st value"))
 f=int(input("Enter the 2nd value"))
 print(mul(e,f))
elif choice ==4:
 g=int(input("Enter the 1st value"))
 h=int(input("Enter the 2nd value"))
 print(areadOfSquare(s))
else:
 print("wrong choice")
```

INPUT AND OUTPUT:

Select operation.
1.Add
2.Subtract
3.Multiply

4.Divide
Enter choice(1/2/3/4):1
Enter first number: 3
Enter second number: 4
3 + 4 = 7

RESULT:

Thus the program has been executed successfully with the desired output. 4

Ex No: 2 Operations on Matrices and Vectors

WRITE THE FOLLOWING PROGRAMS
A) DEFINES A MATRIX AND PRINTS
B) addition of two square matrices
C) MULTIPLICATION OF TWO SQUARE MATRICES

AIM:

Write a program that defines a matrix and prints, addition of two square matrices and multiplication of two square matrices.

PROGRAM:

```
row=int(input("Enter No of Rows for 1st Matrix:"))
column=int(input("Enter No of column for 1nd Matrix:"))
row1=int(input("Enter No of Rows for 2st Matrix:"))
column1=int(input("Enter No of column for 2nd Matrix:"))
X = [[int(input(("Enter value for X[",i,"][",j,"]:")))
for j in range(column)] for i in range(row)]
Y = [[int(input(("Enter value for Y[",i,"][",j,"]:")))
for j in range(column1)] for i in range(row1)]
print("1st Matrix X:",X)
print("2st Matrix Y:",Y)
if row==row1 and column==column1:
   result = [[X[i][j] + Y[i][j]
    for j in range(len(X))]
    for i in range(len(X[0]))]
   print(result)
else:
   print("Adition 2 Matrix not Possible")
for j in range(len(Y[0])):
    for i in range(len(Y)):
       result[i][j] += X[i][j] * Y[i][j]
for r in result:
   print(r)
```

OUTPUT:

```
Enter No of Rows for 1st Matrix:2
Enter No of column for 1nd Matrix:2
Enter No of Rows for 2st Matrix:2
Enter No of column for 2nd Matrix:2
('Enter value for X[', 0, '][', 0, ']:')2
('Enter value for X[', 0, '][', 1, ']:')2
('Enter value for X[', 1, '][', 0, ']:')2
('Enter value for X[', 1, '][', 1, ']:')2
('Enter value for Y[', 0, '][', 0, ']:')2
('Enter value for Y[', 0, '][', 1, ']:')2
('Enter value for Y[', 1, '][', 0, ']:')2
('Enter value for Y[', 1, '][', 1, ']:')2
1st Matrix X: [[2, 2], [2, 2]]
```



```
2st Matrix Y: [[2, 2], [2, 2]]
ADDITION OF TWO SQUARE MATRICES
[[4, 4], [4, 4]]
MULITIPLICATION OF TWO SQUARE MATRICES
[8, 8]
```

[8, 8]

RESULT:

Thus the program has been executed successfully with the desired output.

Ex No: 3 Vectorized operation on simple matrix operations

AIM:

To perform vectorized operation on simple matrix operation in Python program. **PROGRAM:**


```
# importing the modules
import numpy as np
import timeit
```

```
# vectorized sum
print(np.sum(np.arange(15000)))

print("Time taken by vectorized sum : ", end = "")
%timeit np.sum(np.arange(15000))

# iterative sum
total = 0
 for item in range(0, 15000):
    total += item
a = total
print("\n" + str(a))
 print("Time taken by iterative sum : ", end = "")
%timeit a
```

Output:

112492500

Time taken by vectorized sum : 28.5 µs ± 4.34 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

112492500

Time taken by iterative sum : 57.9 ns ± 13.3 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)

RESULT:

Thus the program has been executed successfully with the desired output.

Ex No: 4 Creating Various types of plots /charts from various data source

AIM:

Write a Python program to create various types of plots from various data sources.

PROGRAM:

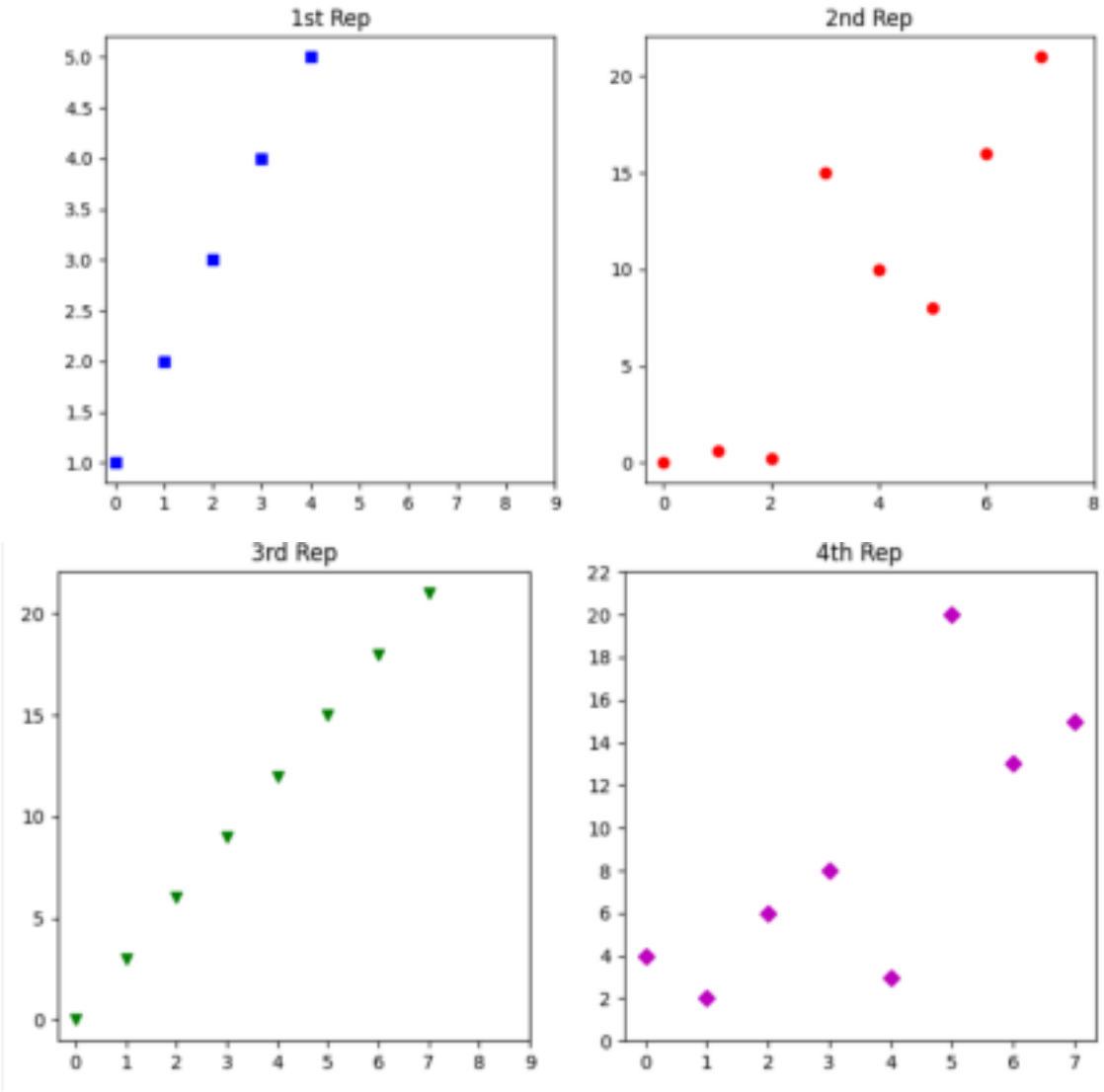
```
import matplotlib.pyplot as plt
a = [1, 2, 3, 4, 5]
```



```
b = [0, 0.6, 0.2, 15, 10, 8, 16, 21]
```

```
c = [4, 2, 6, 8, 3, 20, 13, 15]
# use fig whenever u want the
# output in a new window also
# specify the window size you
# want ans to be displayed
fig = plt.figure(figsize =(10, 10))
# creating multiple plots in a
# single plot
sub1 = plt.subplot(2, 2, 1)
sub2 = plt.subplot(2, 2, 2)
sub3 = plt.subplot(2, 2, 3)
sub4 = plt.subplot(2, 2, 4)
sub1.plot(a, 'sb')
# sets how the display subplot
# x axis values advances by 1
# within the specified range
sub1.set_xticks(list(range(0, 10,
1))) sub1.set_title('1st Rep')
sub2.plot(b, 'or')
# sets how the display subplot x axis
# values advances by 2 within the
# specified range
sub2.set_xticks(list(range(0, 10,
2))) sub2.set_title('2nd Rep')
# can directly pass a list in the plot
# function instead adding the reference
sub3.plot(list(range(0, 22, 3)), 'vg')
sub3.set_xticks(list(range(0, 10,
1))) sub3.set_title('3rd Rep')
sub4.plot(c, 'Dm')
# similarly we can set the ticks for
# the y-axis range(start(inclusive),
# end(exclusive), step)
sub4.set_yticks(list(range(0, 24,
2))) sub4.set_title('4th Rep')
# without writing plt.show() no plot
# will be visible
plt.show()
```

OUTPUT:



RESULT:

Thus the program has been executed successfully with the desired output.

Ex No: 5 Create subplots and color plots

AIM:

To draw basic plots in Python program using Matplotlib.

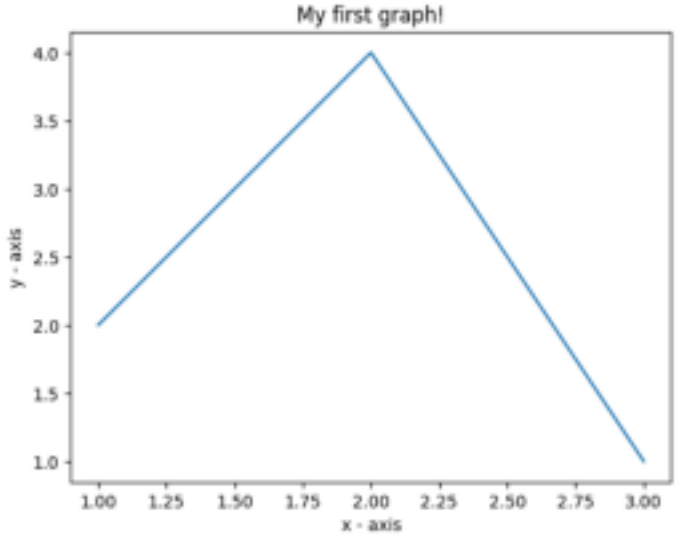
PROGRAM:

```
# importing the required module
import matplotlib.pyplot as plt
# x axis values
```



```
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]
# plotting the points
plt.plot(x, y)
# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')
# giving a title to my graph
plt.title('My first graph!')
# function to show the plot
plt.show()
```

OUTPUT:



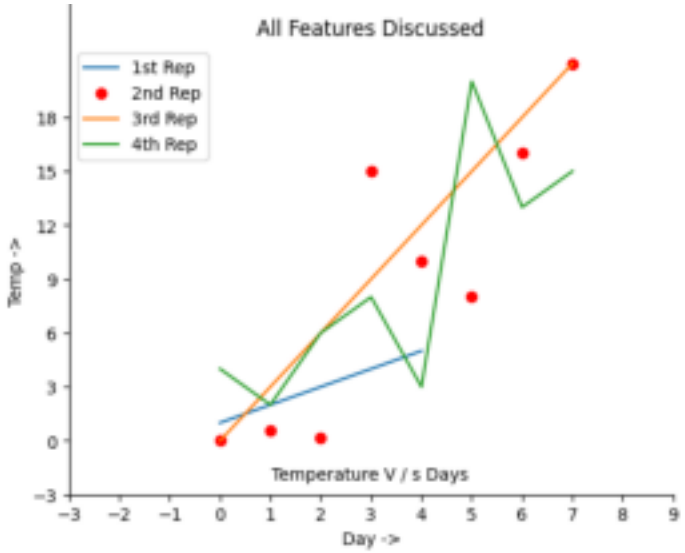
PROGRAM:5B

```
import matplotlib.pyplot as plt
a = [1, 2, 3, 4, 5]
b = [0, 0.6, 0.2, 15, 10, 8, 16, 21]
plt.plot(a)
# o is for circles and r is
# for red
plt.plot(b, "or")
plt.plot(list(range(0, 22, 3)))
# naming the x-axis
plt.xlabel('Day ->')
# naming the y-axis
plt.ylabel('Temp ->')
```



```
c = [4, 2, 6, 8, 3, 20, 13, 15]
plt.plot(c, label = '4th Rep')
# get current axes command
ax = plt.gca()
# get command over the individual
# boundary line of the graph body
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
# set the range or the bounds of
# the left boundary line to fixed range
ax.spines['left'].set_bounds(-3, 40)
# set the interval by which
# the x-axis set the marks
plt.xticks(list(range(-3, 10)))
# set the intervals by which y-axis
# set the marks
plt.yticks(list(range(-3, 20, 3)))
# legend denotes that what color
# signifies what
ax.legend(['1st Rep', '2nd Rep', '3rd Rep', '4th
Rep']) # annotate command helps to write
# ON THE GRAPH any text xy denotes
# the position on the graph
plt.annotate('Temperature V / s Days', xy = (1.01, -2.15))
# gives a title to the Graph
plt.title('All Features Discussed')
plt.show()
```

OUTPUT:



RESULT:

Thus the program has been executed successfully with the desired output.

Ex No: 6 Implement linear regression problem

AIM:

To write a program to implement linear regression problem using python.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as pit
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
x = np.array([1,2,3,4,5])
y = np.array([7,14,15,18,19])
x = x.reshape(-1,1)
regression_model = LinearRegression()
# Fit the data(train the model)
regression_model.fit(x, y)
# Predict
y_predicted = regression_model. predict(x)
#It model evaluation
```



```
mse=mean_squared_error (y, y_predicted)
rmse = np.sqrt(mean_squared_error(y, y_predicted))
```

```
r2 = r2_score(y, y_predicted)
#It printing values
print('Slope:', regression_model.coef_)
print('Intercept:' , regression_model.intercept_)
print('FISE:',mse)
print('Root mean squared error:' , rmse)
print('F2 score:', r2)
```

OUTPUT:

```
Slope: [2.8]
Intercept: 6.199999999999999
FISE: 2.160000000000001
Root mean squared error: 1.4696938456699071
F2 score: 0.8789237668161435
```

RESULT:

Thus the program has been executed successfully with the desired output.

Ex No: 7 Implementation of Linear regression with multiple regression

AIM:

To write a program to implement Linear regression with multiple regression using Python.

PROGRAM:

```
#Import packages and classes
import numpy as np
from sklearn.linear_model import LinearRegression
#Provide data
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
#Create a model and fit it
model = LinearRegression().fit(x, y)
r_sq = model.score(x, y)
#Get results
print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")
new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
print(f"intercept: {new_model.intercept_}")
```

```
print(f"slope: {new_model.coef_}")
#Predict response
y_pred = model.predict(x)
print(f"predicted response:\n{y_pred}")
```

OUTPUT:

```
coefficient of determination: 0.7158756137479542
intercept: 5.633333333333329
slope: [0.54]
intercept: [5.63333333]
slope: [[0.54]]
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333
35.33333333]
```

RESULT:

Thus the program has been executed successfully with the desired output.

Ex No: 8 Implementation of Data preprocessing methods, Correlation matrix

AIM:

To write a Python program to implement Data preprocessing methods, Correlation matrix.

PROGRAM:

Creation of Correlation matrix using Numpy

```
import numpy as np

# x represents the total sale in dollars
x = [215, 325, 185, 332, 406, 522, 412,
     614, 544, 421, 445, 408],

# y represents the temperature on each day of sale
y = [14.2, 16.4, 11.9, 15.2, 18.5, 22.1,
      19.4, 25.1, 23.4, 18.1, 22.6, 17.2]

# create correlation matrix
matrix = np.corrcoef(x, y)
print(matrix)
```

Output:

```
[[1. 0.95750662]
 [0.95750662 1. ]]
```

Creation of Correlation matrix using Pandas

```
import pandas as pd

# collect data
data = {
   'x': [45, 37, 42, 35, 39],
   'y': [38, 31, 26, 28, 33],
   'z': [10, 15, 17, 21, 12]
}

# form dataframe
dataframe = pd.DataFrame(data, columns=['x', 'y', 'z'])
print("Dataframe is : ")
print(dataframe)

# form correlation matrix
matrix = dataframe.corr()
print("Correlation matrix is : ")
print(matrix)
```

OUTPUT:

```
Dataframe is :
   x y z
0 45 38 10
1 37 31 15
2 42 26 17
3 35 28 21
4 39 33 12
Correlation matrix is :
        x y z
x 1.000000 0.518457 -0.701886
y 0.518457 1.000000 -0.860941
z -0.701886 -0.860941 1.000000
```

Correlation matrix using Matplotlib

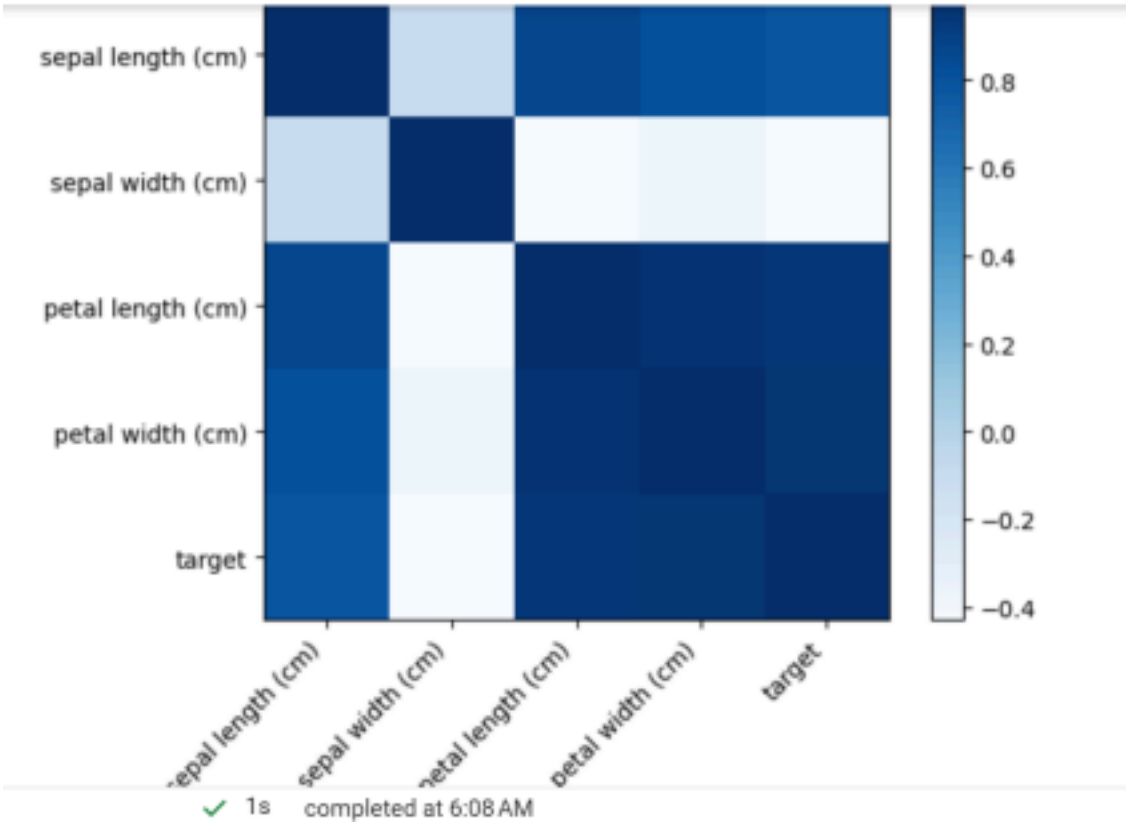
```
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd
dataset = datasets. load_iris ()
dataframe = pd. DataFrame (data = dataset. data, columns = dataset.
feature_names) dataframe ["target"] = dataset. target
matrix = dataframe.corr()

#plotting correlation matrix
plt.imshow(matrix, cmap='Blues')

#adding colorbar
```

```
plt.colorbar()
```



```
#extracting variable names
variables = []
for i in matrix.columns:
variables.append(i)

# Adding labels to the matrix
plt.xticks(range(len(matrix)), variables, rotation=45, ha='right')
plt.yticks(range(len(matrix)), variables)

# Display the plot
plt.show()
```

OUTPUT:



RESULT:

Thus the program has been executed successfully with the desired output. 16

Ex. No: 9 Implementation of spam and non-spam classification problem

AIM:

To write a Python program to implement spam and non-spam classification problem.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm

spam = pd.read_csv('/content/drive/MyDrive/Alumni/spam (1).csv')

z = spam['v2']
y = spam["v1"]
z_train, z_test,y_train, y_test = train_test_split(z,y,test_size = 0.2)

cv = CountVectorizer()
features = cv.fit_transform(z_train)

model = svm.SVC()
model.fit(features,y_train)

features_test = cv.transform(z_test)
print("Accuracy: {}".format(model.score(features_test,y_test)))
```

OUTPUT:

Accuracy: 0.9856502242152466

RESULT:

Thus the program has been executed successfully with the desired output.

Ex. No: 10 Implementation of classifier problem

AIM:

To write a Python program to implement Classifier problem.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm

spam = pd.read_csv('/content/drive/MyDrive/Alumni/spam (1).csv')

z = spam['v2']
y = spam["v1"]
z_train, z_test,y_train, y_test = train_test_split(z,y,test_size = 0.2)

cv = CountVectorizer()
features = cv.fit_transform(z_train)

model = svm.SVC()
model.fit(features,y_train)

features_test = cv.transform(z_test)
print("Accuracy: {}".format(model.score(features_test,y_test)))
```

OUTPUT:

Accuracy: 0.9856502242152466

RESULT:

Thus the program has been executed successfully with the desired output. 18

Ex. No: 11 Implementation of K-Mean Clustering

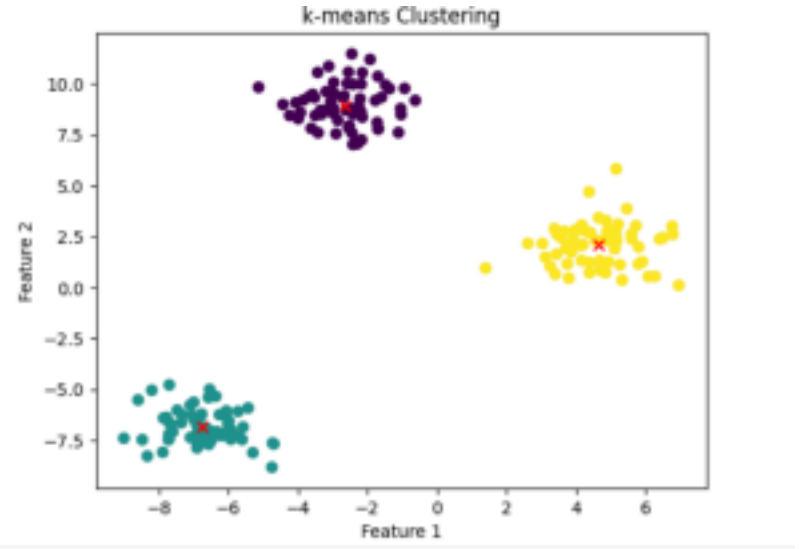
AIM:

To write a Python program to implement K-Mean Clustering algorithm.

PROGRAM:

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
# Generate synthetic data for demonstration
X, y = make_blobs(n_samples=200, centers=3, random_state=42) # Create a
KMeans clustering object and specify the number of clusters kmeans =
KMeans(n_clusters=3) # You can choose the desired number of clusters #
Perform clustering on the data
kmeans.fit(X)
# Get the cluster labels and cluster centers
labels = kmeans.labels_
centers = kmeans.cluster_centers_
# Visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], marker='x', color='red')
plt.title("k-means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

OUTPUT:



RESULT:

Thus the program has been executed successfully with the desired output. **Ex.**

No: 12 : Implementation of K-Mean Clustering

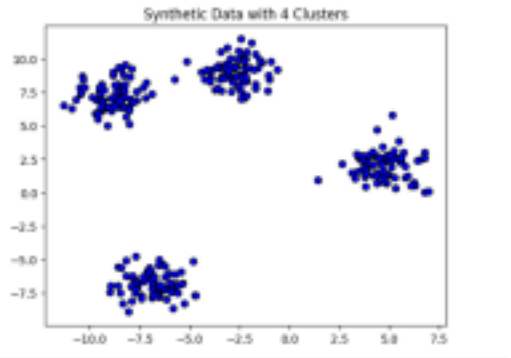
AIM:

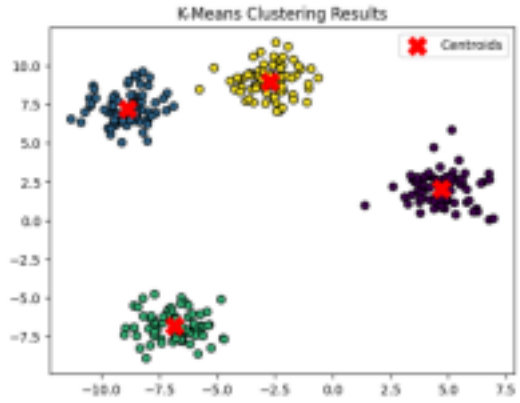
To write a Python program to implement K-Mean Clustering algorithm.

PROGRAM:

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

# Create a synthetic dataset with 4 clusters
X, y = make_blobs(n_samples=300, centers=4, random_state=42, cluster_std=1.0)

# Visualize the data
plt.scatter(X[:, 0], X[:, 1], c='blue', marker='o', edgecolors='k')
plt.title("Synthetic Data with 4 Clusters")
plt.show()
# Apply K-Means clustering with k=4 (number of clusters)
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

# Get the cluster centroids and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
# Visualize the clustered data
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', edgecolors='k')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label='Centroids')
plt.title("K-Means Clustering Results")
plt.legend()
plt.show()
```

OUTPUT:





RESULT:

Thus the program has been executed successfully with the desired output.

Ex. No :13 : Implementation of decision tree

AIM:

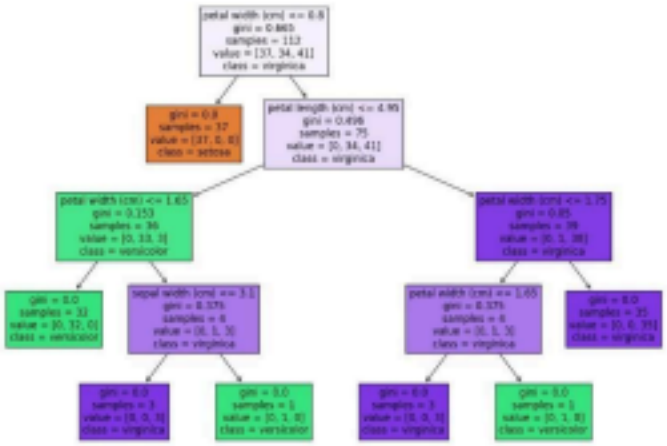
To write a Python program to implement of decision tree algorithm.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data=iris.data,
columns=iris.feature_names) df['target'] = iris.target
X_train, X_test, y_train, y_test = train_test_split(df[iris.feature_names], df['target'],
random_state=0)
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
fig = plt.figure(figsize=(15,10))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
```

OUTPUT:

Accuracy: 0.9736842105263158



RESULT:

Thus the program has been executed successfully with the desired output. **Ex.**

No: 14 Implementation of Random Forest

AIM:

To write a Python program to implement Random Forest algorithm.

PROGRAM:

```
# importing required libraries
# importing Scikit-learn library and datasets package
from sklearn import datasets

# Loading the iris plants dataset (classification)
iris = datasets.load_iris()
# dividing the datasets into two parts i.e. training datasets and test datasets
X, y = datasets.load_iris( return_X_y = True)
  # Splitting arrays or matrices into random train and test subsets
from sklearn.model_selection import train_test_split
# i.e. 70 % training dataset and 30 % test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
# creating dataframe of IRIS dataset
   data = pd.DataFrame({'sepallength': iris.data[:, 0], 'sepalwidth': iris.data[:,
              1], 'petallength': iris.data[:, 2], 'petalwidth': iris.data[:, 3],
                 'species': iris.target})
# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)
  # Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(X_train, y_train)
  # performing predictions on the test dataset
y_pred = clf.predict(X_test)
  # metrics are used to find accuracy or error
from sklearn import metrics
print()

# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test, y_pred))
```

OUTPUT:

```
ACCURACY OF THE MODEL: 0.9333333333333333
```

RESULT:

Thus the program has been executed successfully with the desired output.

Ex. No : 15 Implementation of CART

AIM:

To write a python program to implement CART algorithm.

PROGRAM:

```
import pandas as pd
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```



```
from sklearn import metrics
from sklearn import datasets

#load IRIS dataset
iris = datasets.load_iris()
X = iris.data #load the features
y = iris.target # load target variable

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70%
training and 30% test

# Create Decision Tree classifer object
clf = DecisionTreeClassifier(max_depth=2)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

OUTPUT:

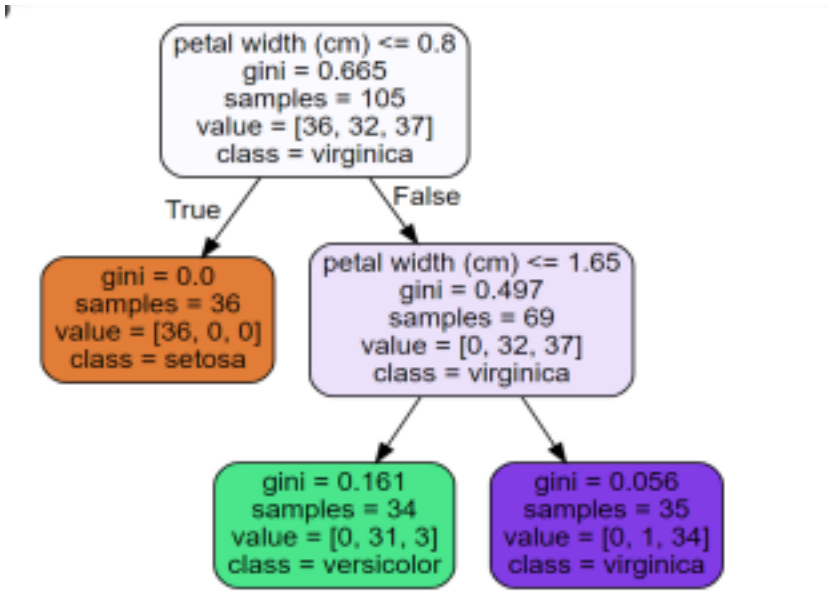
```
Accuracy: 0.9555555555555556
```

PROGRAM:

```
from sklearn.tree import export_graphviz
import os

# Where to save the figures
PROJECT_ROOT_DIR = "."

def image_path(fig_id):
   return os.path.join(PROJECT_ROOT_DIR, "sample_data", fig_id)

export_graphviz(
        clf,
        out_file=image_path("iris_tree.dot"),
        feature_names=iris.feature_names,
        class_names=iris.target_names,
        rounded=True,
        filled=True
   )
```

```
from graphviz import Source
Source.from_file("./sample_data/iris_tree.dot")
```

OUTPUT:



RESULT:

Thus the program has been executed successfully with the desired output.

Content Beyond the Syllabus

1) Implement the principal component analysis algorithm

PROGRAM:

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
# Load the Iris dataset for demonstration
data = load_iris()
X = data.data
y = data.target
```

```
# Create a PCA object and specify the number of components
pca = PCA(n_components=2) # You can choose the desired number of components
# Apply PCA to the data
X_pca = pca.fit_transform(X)
```



```
# Access the principal components
principal_components = pca.components_
# Access the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
# Print the results
print("Principal Components:")
print(principal_components)
print("Explained Variance Ratio:")
print(explained_variance_ratio)
```

OUTPUT:

```
Principal Components:
[[ 0.36138659 -0.08452251 0.85667061 0.3582892 ]
 [ 0.65658877 0.73016143 -0.17337266 -0.07548102]]
Explained Variance Ratio:
[0.92461872 0.05306648]
```

2) Implement the singular value decomposition algorithm.

PROGRAM:

```
import numpy as np
# Create a sample matrix
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Perform Singular Value Decomposition
U, S, VT = np.linalg.svd(A)
# Print the singular values
print("Singular values:")
print(S)
# Print the left singular vectors (U)
```

```
print("Left singular vectors (U):")
print(U)
# Print the right singular vectors (VT)
print("Right singular vectors (VT):")
print(VT)
```


OUTPUT:

```
Singular values:
[1.68481034e+01 1.06836951e+00 4.41842475e-16]
Left singular vectors (U):
[[-0.21483724 0.88723069 0.40824829]
 [-0.52058739 0.24964395 -0.81649658]
 [-0.82633754 -0.38794278 0.40824829]]
Right singular vectors (VT):
[[-0.47967118 -0.57236779 -0.66506441]
 [-0.77669099 -0.07568647 0.62531805]
 [-0.40824829 0.81649658 -0.40824829]]
```

3 Implement the support vector machine algorithm

PROGRAM:

```
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Generate synthetic data for demonstration
X, y = make_classification(n_samples=100, n_features=10, random_state=42)
# Split the data 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)
```

```
# Create an SVM classifier
svm = SVC(kernel='linear') # You can choose different kernel functions like
'linear', 'rbf', 'poly', etc.
# Train the classifier
svm.fit(X_train, y_train)
# Make predictions on the test set
y_pred = svm.predict(X_test)
# Evaluate the classifier
```



```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy: 0.9
```