Objective:

Study of the course enables the learners to make use of the machine learning concepts and algorithms to derive data insights. The course provides exposure to the design and implementation aspects of machine learning algorithms such as decision trees, regression, naive bayes algorithm, clustering algorithms and artificial neural network. This helps the students to develop machine learning based solutions to real world problems.

List of experiments:

1. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

2. Write a program to implement the naïve bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

3. Assuming a set of documents that need to be classified, use the naive bayesian classifier model to perform this task. Calculate the accuracy, precision, and recall for your data set.

4. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set. You can use Python ML library classes/API.

5. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the K-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.

6. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Python ML library classes can be used for this problem.

7. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

8. Write a program to implement 5-fold cross validation on a given dataset. Compare the accuracy, precision, recall, and F-score for your data set for different folds.

9. Implement SVM/Softmax classifier for CIFAR-10 dataset: (i) using KNN, (i) using 3 layer neural network.

10. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

EXPERIMENT - 1

OBJECTIVE

A program to demonstrate the working of the decision tree based DI3 algorithm

ALGORITHM

1. Import all required modules.
2. Read the dataset using pd.read_csv.
3. Split the dataset into 2, one having only target variable and the other having all the features except the target.
4. Split the dataset into test and train set with test size as 0.3
5. Call the decision tree classifier model.
6. Fit the model with the train set using the fit function.
7. predict and test the model using the predict function with a test set.
8. Compute the performance measures using appropriate functions from sklearn.
9. Visualize the decision tree using the create_png function.

DESCRIPTION

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
Invented by Ross Quinlan, ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach mean: that at each iteration we select the best feature at the present moment to create a node.
Most generally ID3 is only used for classification problems with nominal features only.

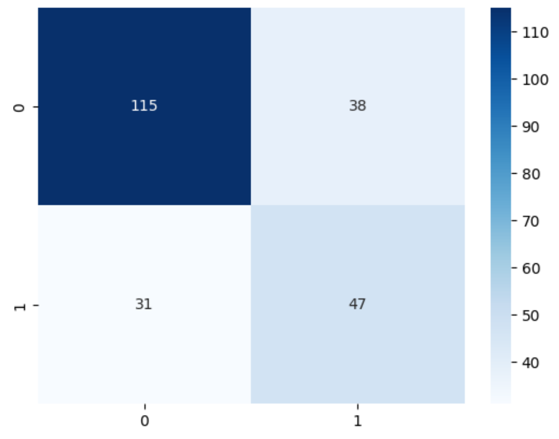
IMPLEMENTATION

```
# importing of necessary modules

import pandas as pd
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier

# select columns for features and target variable
pima = pd.read_csv("/content/diabetes.csv")
selected_columns = ['Pregnancies',   'Glucose','BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age']
X = pima[selected_columns]
y = pima['Outcome']

#split the data into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)

#fit the model with training data
#predict the target values for the test set

dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
y_predict = dtree.predict(X_test)

#calculate and performance matrix
print("Precision = ", metrics.precision_score(y_predict, y_test))
print("Recall = " ,metrics.recall_score(y_predict, y_test))
print("Accuracy = ", metrics.accuracy_score(y_predict, y_test))
print("F1-score = ", metrics.f1_score(y_predict, y_test))
```

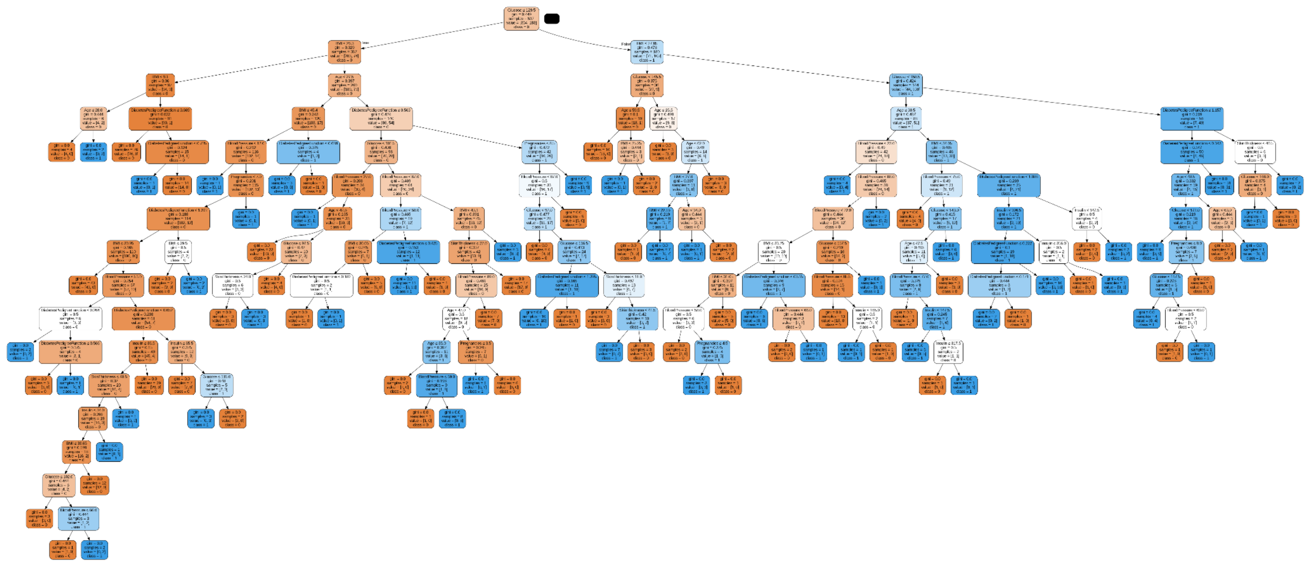
```
Precision =  0.5529411764705883
Recall =  0.6025641025641025
Accuracy =  0.7012987012987013
F1-score =  0.5766871165644172
```

```
#create confusion. matrix and visualize using heatmap
cf_matrix = metrics.confusion_matrix(y_predict, y_test)
print(cf_matrix)
sns.heatmap(cf_matrix, annot = True, cmap = 'Blues', fmt = 'g')
```

```
[[115  38]
 [ 31  47]]
```



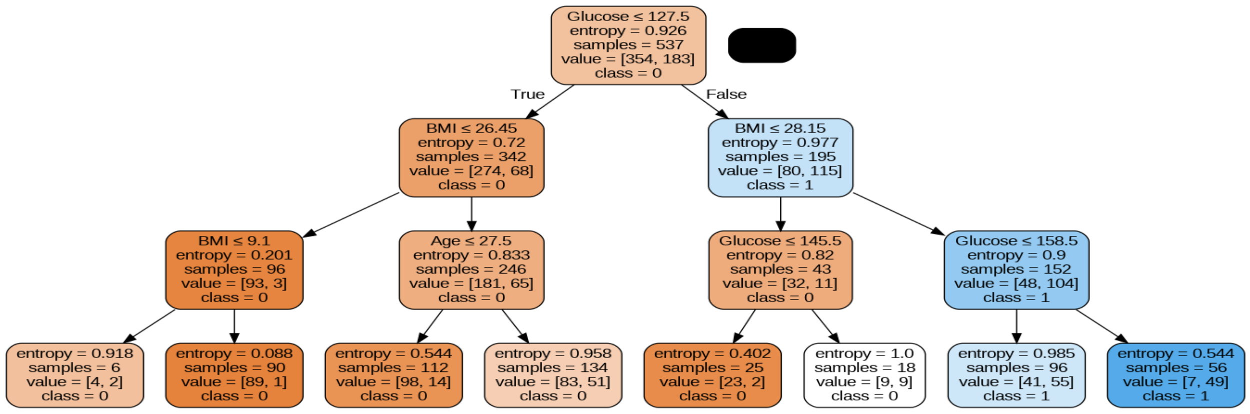
```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

#visualize decision tree and export it as png
dot_data = StringIO()
export_graphviz(dtree, out_file = dot_data, filled = True, rounded = True, special_characters =
True, feature_names = selected_columns, class_names = ['0', '1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write('diabetes.png')
Image(graph.create_png())
```



```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

dtree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 3)
dtree.fit(X_train, y_train)
y_predict = dtree.predict(X_test)

print("Precision = ", metrics.precision_score(y_predict, y_test))
print("Recall = ", metrics.recall_score(y_predict, y_test))
print("Accuracy = ", metrics.accuracy_score(y_predict, y_test))
print("F1-score = ", metrics.f1_score(y_predict, y_test))

from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(dtree, out_file = dot_data, filled = True, rounded = True, special_characters =
True, feature_names = selected_columns, class_names = ['0', '1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write('diabetes.png')
Image(graph.create_png())

import numpy as np
# input = (0,162,76,56,100,53.2,0.759,25)
input = (1,85,66,29,0,26.6,0.351,31)
# 15,136,70,32,110,37.1,0.153,43 - diabetic
# 5,137,108,0,0,48.8,0.227,37 - diabetic
input_array = np.asarray(input)
reshaped_array = input_array.reshape(1,-1)
prediction = dtree.predict(reshaped_array)
if(prediction[0]== 1):
  print('Diabetic \n')
else:
  print("Not diabetic")
```

ANALYSIS

```
Precision =  0.6352941176470588
Recall =  0.7105263157894737
Accuracy =  0.7705627705627706
F1-score =  0.6708074534161491
```



[redacted]

```
Not diabetic
```

EXPERIMENT - 2

OBJECTIVE

Write a program to implement the naive bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

ALGORITHM

1. Import all required modules.
2. Read the dataset using pd.read_csv.
3. Split the dataset into 2 one having only target variables and the other having all the features except target.
4. Split the dataset into test and train set with test size as 0.2
5. Call the naive bayes gaussian classifier model.
6. Fit the model with the train set using the fit function.
7. predict and test the model using the predict function with the test set.
8. Compute the performance measures using appropriate functions from sklearn.

DESCRIPTION

Naive Bayes algorithm is a supervised classification algorithm based on Bayes theorem and used for prediction on the basis of probability of an object. Its called naive because it assumes that the occurrence of a certain feature is independent of the occurrence of the features.There are three types of Naive Bayes models. They are :

1. Gaussian model : Assume that features follow a normal distribution.
2. Multinomial : Used when data is multi normally distributed.
3. Bernoulli : Similar to multinomial, but the predictor variable are independent boolean variables.

IMPLEMENTATION

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
ds = pd.read_csv('/content/heart.csv')
x = ds.drop(columns = 'target', axis =1)
y= ds['target']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=1)
print(x.shape, x_train.shape, x_test.shape)
nb = GaussianNB()
nb.fit(x_train, y_train)
training_prediction = nb.predict(x_train)
training_accuracy = metrics.accuracy_score(training_prediction, y_train)
print('training accuracy = ',training_accuracy)
testing_prediction = nb.predict(x_test)
testing_accuracy = metrics.accuracy_score(testing_prediction, y_test)
print('testing accuracy = ',testing_accuracy)
input = (62,1,0,120,267,0,1,99,1,1.8,1,2,3)
input_array = np.asarray(input)
reshape_array = input_array.reshape(1,-1)
prediction = nb.predict(reshape_array)
if prediction[0]==0:
  print('No')
Else:
 print('Yes')
```

OUTPUT

```
(1025, 13) (820, 13) (205, 13)
training accuracy =  0.7804878048780488
testing accuracy =  0.8426829268292683
No
```

ANALYSIS

The GaussianNB classifier has an training accuracy of 0.78 on the UCI heart disease dataset and an accuracy of 0.84

EXPERIMENT - 3

OBJECTIVE

Assuming a set of documents that need to be classified, use the naive bayesian classifier model to perform the task.

ALGORITHM

1. Import all required modules.
2. Read the dataset using pd.read_csv.
3. Split the dataset into 2- one having only the target variable and the other having all the features except the target.
4. Split the dataset into a test and train set.
5. Call the naive bayes classifier model.
6. Fit the model with the train set using the fit function.
7. predict and test the model using the predict function with the test set.
8. Compute the performance measures using appropriate functions.

DESCRIPTION

The Naive Bayes classification algorithm is a probabilistic classifier. It is based on probability models that incorporate strong independence assumptions. The independence assumptions often do not have an impact on reality. Therefore they are considered naive.

IMPLEMENTATION

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
ds = fetch_20newsgroups()
ds.target_names
cat = ['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
train = fetch_20newsgroups(subset= 'train', categories = cat )
test =  fetch_20newsgroups(subset= 'test', categories = cat )
print(len(test.data))
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn import metrics
model = make_pipeline(TfidfVectorizer(),MultinomialNB())
model.fit(train.data, train.target)
labels = model.predict(test.data)
print("Precision = ", metrics.precision_score(test.target, labels, average='weighted'))
print("Accuracy = ", metrics.accuracy_score(test.target, labels))
print("Recall =", metrics.recall_score(test.target,labels, average='weighted'))
print("F1-score =", metrics.f1_score(test.target,labels, average='weighted'))
```

OUTPUT

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'Talk.religion.misc']


7532

Precision =  0.8218781741893993
Accuracy =  0.7738980350504514
Recall = 0.7738980350504514
F1-score = 0.7684457156894653
```

ANALYSIS

The code uses a Multinomial Naive Bayes classifier on the 20 newsgroups dataset, achieving reasonably high accuracy and precision scores, suggesting effective text classification based on TF-IDF vectorization.

EXPERIMENT - 4

OBJECTIVE

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML.

ALGORITHM

1. Import required modules
2 .Read the dataset
3. Clean the dataset
4. Call BayesianModel
5. Learn CPD using Maximum likelihood estimators
6. Inference with Bayesian Network
7. Calculate the probability

DESCRIPTION

Bayesian network (also known as a Bayes network, Bayes net, belief network, or decision network) is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). Bayesian networks are ideal for taking an —----- that occurred and predicting the likelihood that any one of several possible known causes was the contributing factor. It satisfies the local Markov property: each variable is conditionally independent of its non-descendants given its parent variables. Developing a Bayesian network often begins with creating a DAG G such that X satisfies the local Markov property with respect to G.

IMPLEMENTATION

```
pip install pgmpy

import numpy as np
import pandas as pd
import csv
import pgmpy
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination
import seaborn as sns
df = pd.read_csv('/content/heart.csv')
df = df.replace('?',np.nan)
model = BayesianNetwork([('oldpeak','target'),('ca','target'),('exang','target'),('thal','target'),
                    ('restecg','target'),('target','chol')])
print('\n Learning CPD Using MaximumLikelihoodEstimator')
model.fit(df, estimator = MaximumLikelihoodEstimator)
print('\n Inferencing with bayesian network')
heart_infer = VariableElimination(model)
print('\n Probability of  heart disease given evidence = restecg')
q1 = heart_infer.query(variables = ['target'], evidence = {'restecg':1})
print(q1)
print('\n Probability of  heart disease given evidence = thal')
q2 = heart_infer.query(variables = ['target'], evidence = {'thal':3})
print(q2)
```

OUTPUT

```
 Learning CPD Using MaximumLikelihoodEstimator
 Inferencing with bayesian network
 Probability of heart disease given evidence = restecg
+-----------+---------------+
|   target  |   phi(target) |
+===========+===============+
| target(0) |        0.4141 |
+-----------+---------------+
| target(1) |        0.5859 |
+-----------+---------------+

 Probability of heart disease given evidence = thal
+-----------+---------------+
|    target |   phi(target) |
+===========+===============+
| target(0) |        0.5306 |
+-----------+---------------+
| target(1) |        0.4694 |
+-----------+---------------+
```

ANALYSIS

Probability of having heart disease given restecg is 1 is `0.4141` and probability of not having heart disease given restecg is 1 is `0.5859`

Probability of having heart disease given thal(A blood disorder called thalassemia (3 = normal; 6 = fixed defect; 7 = reversible defect)) is 3 is `0.4694` and probability of not having heart disease given thal is 3 is `0.5306`

EXPERIMENT - 5

OBJECTIVE

Apply EM algorithm to cluster a set of data stored in a .CSV fie. Use the same data set for clustering using the k- Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML Library classes/API in the program.

ALGORITHM

1. Import all required modules.
2. Read the dataset using pd.read_csv.
3. Call KMeans model with 3 clusters and fit model with training dataset.
4. Find the accuracy of the K Means model.
5. Call a Gaussian Mixture model with 3 components and fit a model with a training dataset.
6. Find the accuracy of the K Means model.
7. Compare the accuracy.

DESCRIPTION

K-Means Algorithm
K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters.
Expectation Maximization Clustering
EM is an iterative method which alternates between two steps, expectation (E) and maximization (M). For clustering, EM makes use of the finite Gaussian mixtures model and estimates a set of parameters iteratively until a desired convergence value is achieved.

IMPLEMENTATION

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
import sklearn.metrics as sm
import pandas as pd
import numpy as np
%matplotlib inline
```

```
df = pd.read_csv("/content/Iris.csv")
X = df.drop(columns = ['Species'])
Y = df.drop(columns = ['SepalLengthCm','SepalWidthCm',  'PetalLengthCm', 'PetalWidthCm',
'Species'])
```

```
model = KMeans(n_clusters = 3)
model.fit(X)
preds = model.predict(X)
preds
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

```
sm.accuracy_score(Y, model.labels_)*100
```

```
0.006666666666666667
```

```
cm = confusion_matrix(Y, model.labels_)
print(cm)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]]
```

```
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xx = scaler.transform(X)
Xx = pd.DataFrame(xx, columns = X.columns)
X= Xx
```

```
gmm = GaussianMixture(n_components = 3)
gmm.fit(X)
y_cluster_gmm = gmm.predict(X)
y_cluster_gmm
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
sm.accuracy_score(Y, y_cluster_gmm)
```

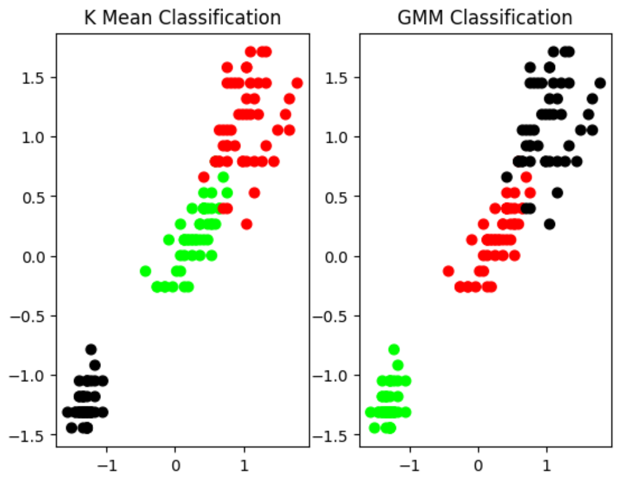
```
0.006666666666666667
```

```
cm = confusion_matrix(Y, y_cluster_gmm)
print(cm)
```

```
[[0 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 ...
 [0 0 1 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 1 ... 0 0 0]]
```

```
plt.figure(figsize = (14, 7))
colormap = np.array(['red', 'lime', 'black'])

plt.subplot(1,2,1)
plt.scatter(X.PetalLengthCm, X.PetalWidthCm, c = colormap[preds], s = 40)
plt.title('K Mean Classification')
plt.subplot(1,2,2)
plt.scatter(X.PetalLengthCm, X.PetalWidthCm, c = colormap[y_cluster_gmm], s = 40)
plt.title('GMM Classification')
```

ANALYSIS



For Iris dataset, GMM performs better than K-means for classification.

EXPERIMENT - 6

OBJECTIVE

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Python ML library classes can be used for this problem.

ALGORITHM

1. Import all required modules.
2. Read the dataset from sklearn dataset.
3. Extract independent variables.
4. Split the dataset into a test and train set with test size 30%.
5. Train KNN algorithm in the dataset.
6. Test the model with a test set.
7. Compute the accuracy using the appropriation function from sklearn.
8. Print correct and wrong predictions along with the true labels.

DESCRIPTION

The k-nearest neighbors algorithm, also known as KNN or k-NN is a non-parametric,supervised learning classifier which uses proximity to make classifications or predictions about the grouping of an individual data point.

IMPLEMENTATION

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

iris_data = datasets.load_iris()
iris_df = pd.DataFrame(data=np.c_[iris_data['data'],iris_data['target']],columns=
iris_data['feature_names']+['target'])
iris_df

X=iris_df.drop(columns=['target'])
y=iris_df['target']
X_train,X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=1)

knn = KNeighborsClassifier(n_neighbors=6)
knn = knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)

print("Accuracy of model:", accuracy_score(y_pred,y_test))

i=0
print('Actual label        Predicted Label          Correct/wrong prediction')
for label in y_test:
  print('%-25s %-25s' %(label,y_pred[i]), end="")
  if (label== y_pred[i]):
        print('%-25s' %('correct'));
  else:
        print('%-25s' % ('Wrong'));
  i=i+1
```

OUTPUT

```
Accuracy of model: 0.9777777777777777

Actual label      Predicted Label      Correct/wrong prediction
0.0                        0.0                       correct
1.0                        1.0                       correct
1.0                        1.0                       correct
0.0                        0.0                       correct
2.0                        2.0                       correct
```

```
1.0                         1.0                        correct
2.0                         2.0                        correct
0.0                         0.0                        correct
0.0                         0.0                        correct
2.0                         2.0                        correct
1.0                         1.0                        correct
0.0                         0.0                        correct
2.0                         2.0                        correct
1.0                         1.0                        correct
1.0                         1.0                        correct
0.0                         0.0                        correct
1.0                         1.0                        correct
1.0                         1.0                        correct
0.0                         0.0                        correct
0.0                         0.0                        correct
1.0                         1.0                        correct
1.0                         1.0                        correct
1.0                         1.0                        correct
0.0                         0.0                        correct
2.0                         2.0                        correct
1.0                         1.0                        correct
0.0                         0.0                        correct
0.0                         0.0                        correct
1.0                         1.0                        correct
2.0                         2.0                        correct
1.0                         1.0                        correct
2.0                         2.0                        correct
1.0                         1.0                        correct
2.0                         2.0                        correct
2.0                         2.0                        correct
0.0                         0.0                        correct
1.0                         1.0                        correct
0.0                         0.0                        correct
1.0                         1.0                        correct
2.0                         2.0                        correct
2.0                         2.0                        correct
0.0                         0.0                        correct
2.0                         1.0                        Wrong
2.0                         2.0                        correct
1.0                         1.0                        correct
```

ANALYSIS

The accuracy of the KNN model on the iris dataset is found to be 97% with 30% test size and number of neighbors as 6.

EXPERIMENT - 7

OBJECTIVE

Implement the nonparametric Locally weighted Regression algorithm inorder to fit data points. Select the appropriate dataset for your experiment and draw graphs.

ALGORITHM

1. Import required modules
2. Read the dataset
3. Determine the weights and bias parameters such that it minimizes residual sum of weighted squared errors.
4. Predict Xoβ to fit a smooth curve through points in a scatter plot.

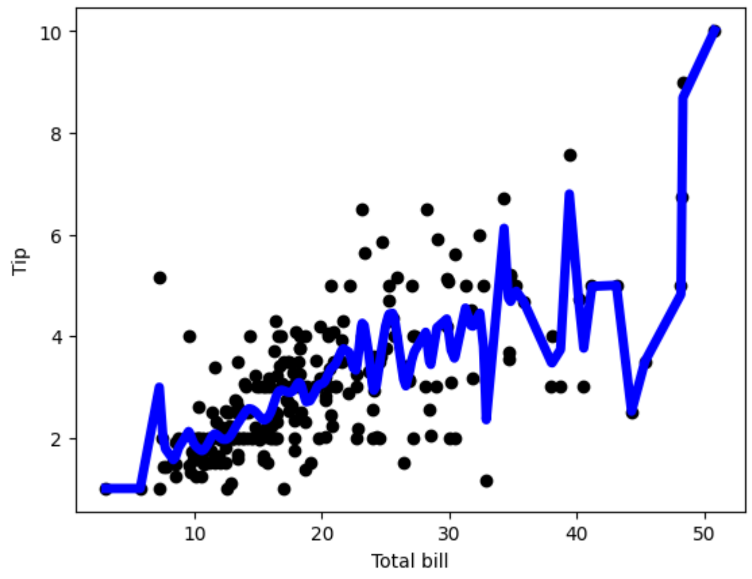
DESCRIPTION

Locally weighted regression is a very powerful non parametric model used in statistical learning that is the model does not learn a fixed set of parameters as is done in ordinary linear regression rather parameter ‘θ’ is computed individually for each query point x. While computing 0, a higher “preference” is given to the points in the training set lying vertically of x than the points lying far away from x. Given a dataset x,y we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors. The weights are given by a kernel function (k or w) which can be chosen arbitrarily.

IMPLEMENTATION

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
def Kernel(point,xmat,k):
m,n=np.shape(xmat)
weights=np.mat(np.eye((m)))
for j in range(m):
diff=point-X[j]
weights[j,j]=np.exp(diff*diff.T/(-2.0*k*k**2))
return weights
def LW(point,xmat,ymat,k):
wei=Kernel(point,xmat,k)
W=(X.T*(wei*X)).I*(X.T*(wei*ymat.T))
return W
def LWR(xmat,ymat,k):
m,n=np.shape(xmat)
ypred=np.zeros(m)
for i in range(m):
ypred[i]=xmat[i]*LW(xmat[i], xmat,ymat,k)
return ypred
data=pd.read_csv('tips.csv')
bill=np.array(data.total_bill)
tip=np.array(data.tip)
mbill=np.mat(bill)
mtip=np.mat(tip)
m=np.shape(mbill)[1]
one=np.mat(np.ones(m))
X=np.hstack((one.T,mbill.T))
ypred=LWR(X,mtip,0.5)
SortIndex=X[:,1].argsort(0)
xsort=X[SortIndex][:,0]
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='black')
ax.plot(xsort[:,1],ypred[SortIndex],color='blue',linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

OUTPUT:



ANALYSIS

The data points are fit using non parametric locally weighted regression.

EXPERIMENT - 8

OBJECTIVE

Write a program to implement 5 fold cross validation on a given dataset.Compare the accuracy,precision,recall, and F-score for your dataset for different folds.

ALGORITHM

Step 1 : Import all required libraries.
Step 2 : Read the dataset from sklearn dataset library.
Step 3 : Extract independent variables.
Step 4 : Train the decision tree classifier on the dataset.
Step 5 : Compute the train and validation performance measures using cross validation function.
Step 6 : compare the accuracy, precision,recall,and E-score for different folds using bar plots.

DESCRIPTION

Cross-validation is a statistical method used to estimate the skill of machine learning models. k-fold cross-validation is when the dataset is split into a K number of folds and is used to evaluate the model's ability when given new data. k refers to the number of groups the data sample is split into. For example, if you see that the k-value is 5, we can call this a 5-fold cross-validation.

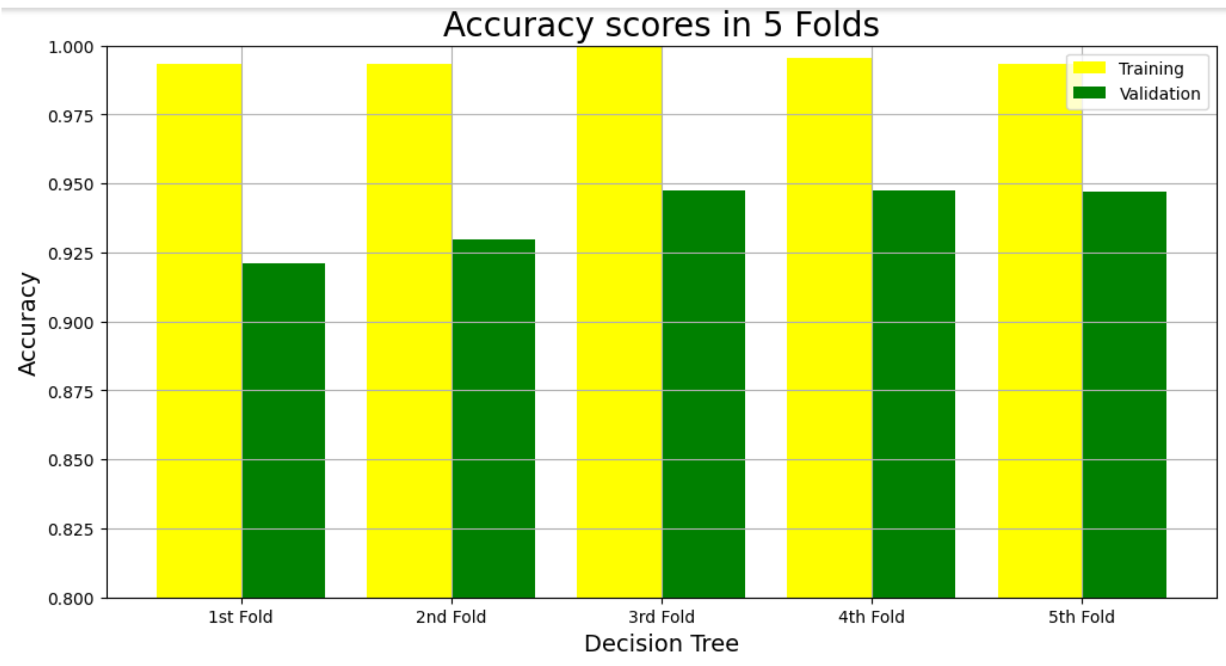
IMPLEMENTATION

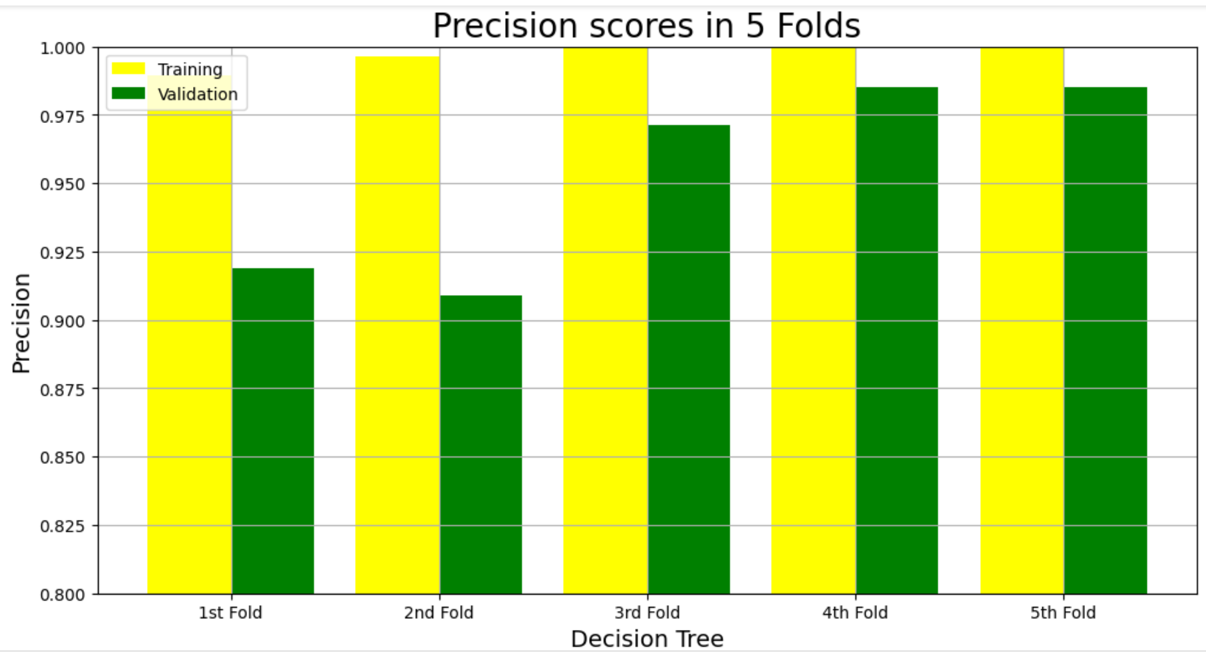
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_validate
cancer= datasets.load_breast_cancer()
df=pd.DataFrame(cancer.data, columns=cancer.feature_names)
df['target'] = pd.Series(cancer.target)
df.head(3)
x=df.drop(columns=['target'])
y=df['target']
def cross_validation (model, _x, _y, _cv=5):

 _scoring = ['accuracy', 'precision', 'recall', 'f1']

 results = cross_validate(estimator=model,
                          X=_x,
                          y=_y,
                          cv=_cv,
                          scoring = _scoring,
                          return_train_score=True)
print("Mean Training Accuracy scores: ",results['train_accuracy'].mean())

 print("Mean Training Precision scores: ",results['train_precision'].mean())

 print("Mean Training Recall scores: ",results['train_recall'].mean())

 print("Mean Training F1 scores: ",results['train_f1'].mean())

 print("Mean Validation Accuracy scores: ",results['test_accuracy'].mean())

 print("Mean Validation Precision scores: ",results['test_precision'].mean())

 print("Mean Validation Recall scores: ",results['test_recall'].mean())

 print("Mean Validation F1 scores: ",results['test_f1'].mean())
 return   { "Training Accuracy scores": results['train_accuracy'],
            "Training Precision scores": results['train_precision'],

            "Training Recall scores": results['train_recall'],

            "Training F1 scores": results['train_f1'],

            "Validation Accuracy scores": results['test_accuracy'],

            "Validation Precision scores": results['test_precision'],

            "Validation Recall scores": results['test_recall'],

            "Validation F1 scores": results['test_f1'],

}
from sklearn.tree import DecisionTreeClassifier

decision_tree_model = DecisionTreeClassifier(criterion="entropy",
min_samples_split=5,random_state=0)
decision_tree_result = cross_validation(decision_tree_model, x, y, 5)
def plot_result(x_label, y_label, plot_title, train_data, val_data):

  plt.figure(figsize=(12,6))

  labels = ["1st Fold", "2nd Fold", "3rd Fold", "4th Fold", "5th Fold"]

  X_axis = np.arange(len(labels))

  ax = plt.gca()

  plt.ylim(0.8, 1)

  plt.bar(X_axis-0.2, train_data, 0.4, color='yellow', label="Training")

  plt.bar(X_axis+0.2, val_data, 0.4, color="green", label='Validation')

  plt.title(plot_title, fontsize=20)

  plt.xticks(X_axis, labels)

  plt.xlabel(x_label, fontsize=14)
  plt.ylabel(y_label, fontsize=14)

  plt.legend()

  plt.grid(True)
model_name = "Decision Tree"
plot_result(model_name, "Accuracy", "Accuracy scores in 5 Folds",
            decision_tree_result["Training Accuracy scores"],
            decision_tree_result["Validation Accuracy scores"])
plt.show()
model_name = "Decision Tree"

plot_result(model_name, "Precision","Precision scores in 5 Folds",
decision_tree_result["Training Precision scores"],
            decision_tree_result["Validation Precision scores"])
plt.show()
model_name= "Decision Tree"
plot_result(model_name, "Recall", "Recall scores in 5 Folds", decision_tree_result["Training
Recall scores"],
            decision_tree_result["Validation Recall scores"])
plt.show()
model_name = "Decision Tree"
plot_result(model_name, "F1-score", "F1-score scores in 5 Folds",
decision_tree_result["Training F1 scores"],
            decision_tree_result["Validation F1 scores"])
plt.show()
```

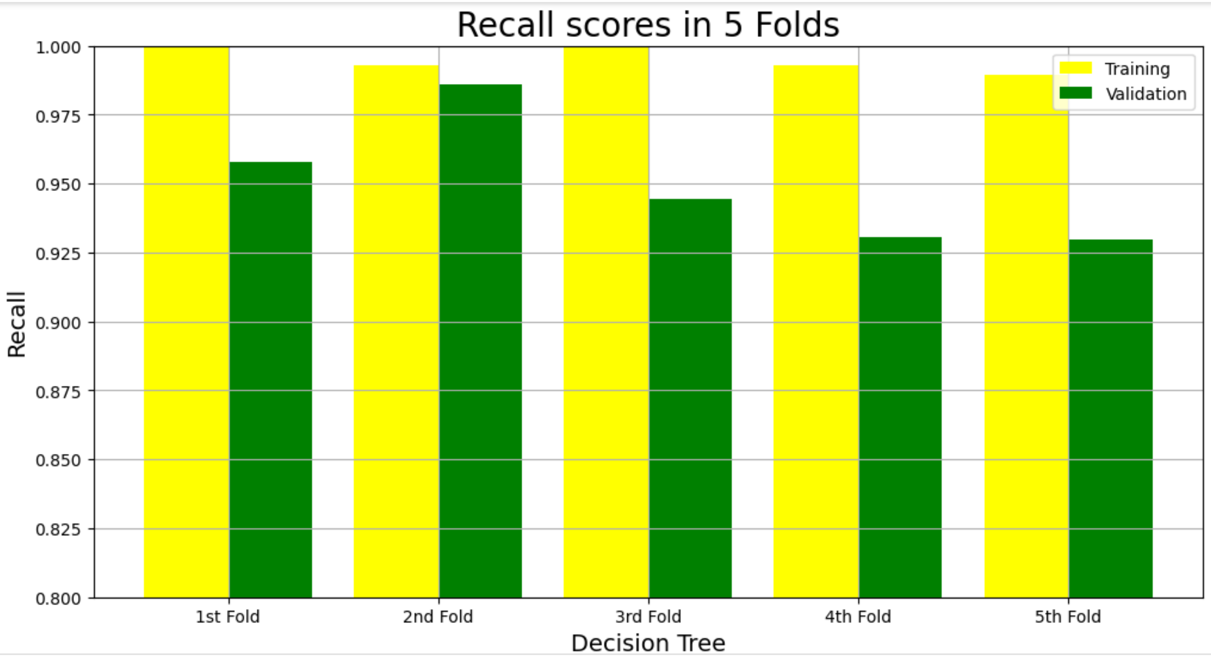
ANALYSIS

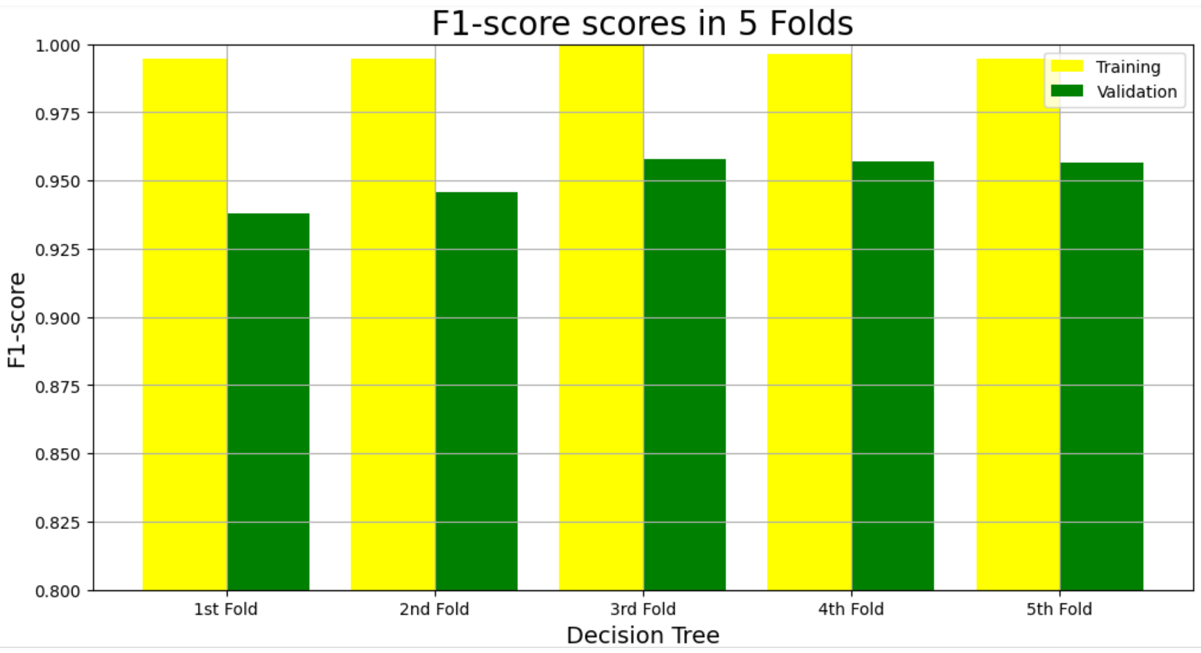
Performance measures upon performing 5-fold Cross validation on breast cancer dataset are as follows:

```
Mean Training Accuracy scores:  0.9951677270098322
Mean Training Precision scores:  0.997222121046561
Mean Training Recall scores:  0.9950999877315667
Mean Training F1 scores:  0.9961470267493764
Mean Validation Accuracy scores:  0.9385033379909953
Mean Validation Precision scores:  0.953961428790226
Mean Validation Recall scores:  0.9496478873239436
Mean Validation F1 scores:  0.9510576111150473
```


Accuracy scores in 5 Folds
Accuracy
Decision Tree
Training
Validation
1st Fold
2nd Fold
3rd Fold
4th Fold
5th Fold
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800


Precision scores in 5 Folds
Precision
Decision Tree
Training
Validation
1st Fold
2nd Fold
3rd Fold
4th Fold
5th Fold
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800


Recall scores in 5 Folds
Recall
Decision Tree
1st Fold
2nd Fold
3rd Fold
4th Fold
5th Fold
Training
Validation
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800


F1-score scores in 5 Folds
F1-score
Decision Tree
1st Fold
2nd Fold
3rd Fold
4th Fold
5th Fold
Training
Validation
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800