Université Jean Monnet

Machine Learning

Practical Session-1 Report

Decision Trees and Neural Networks

Team Members:
Karthik Bhaskar
Rohil Gupta

Supervisor:
Rémi Emonet

March 4, 2018

Contents

1 Introduction **2**

2 Exercise 1: Warm up, visualizing IRIS (Fisher, 1936) **2**

3 Exercise 2: Create your own dataset **9**

4 Exercise 3: KNN classifier on IRIS **11**

5 Exercise 4: Decision Trees on IRIS **16**

6 Exercise 5: Neural Networks on DIGITS **23**

7 Exercise 6: Play with your own data **27**

1 Introduction

The goal of this practical session-1 was to understand the decision trees, neural networks and their learning algorithm.
Scikit-learn was used in this practical session, which is a free software machine learning library for the Python programming language.

This report contains the commands executed for different exercises in the practical session and their answers.

2 Exercise 1: Warm up, visualizing IRIS (Fisher, 1936)

Iris database is a famous database to be found in the field of pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.
The commands executed in this exercise are as follows,

```
from sklearn.datasets import load_iris
irisData = load_iris()
```

Here the Iris dataset is loaded.

```
print(len(irisData.data)) #[1]
```

Output: 150

Answer: 150 is the total number of instances, where each class has 50 instances and there are 3 classes.

```
print(irisData.target_names[0]) #[2]
```

Output: setosa

Answer: Setosa is the first class of the Iris Dataset stored in target_names[0].

```
print(irisData.target_names[-1+len(irisData.target_names)]) #[3]
```

Output: virginica

Answer: The output of len(irisData.target_names) is 3 therefore -1+3 is 2 and "virginica" class is in the 2nd index of the target_names.

```
print(irisData.data.shape) #[4]
```

Output:(150, 4)

Answer: It returns the dimensions of the dataset.

```
print(irisData.data.shape[0]) #[5]
```

Output: 150

Answer: print(irisData.data.shape[0]) displays the number of rows(Instances) in the dataset, which is 150.

```
print(irisData.data[0]) #[6]
```

Output: [5.1 3.5 1.4 0.2]

Answer: This command displays the first row of the dataset.

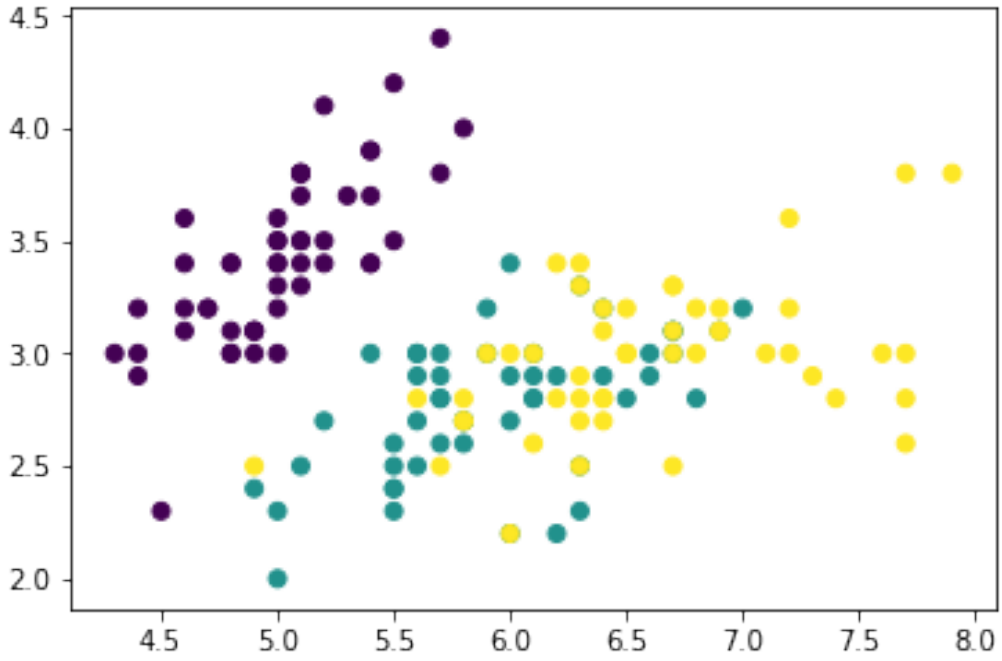
```
plt.scatter(X[:,x], X[:,y], c=Y) #[7]
```

Output: <matplotlib.collections.PathCollection at 0x11414c5f8>

Answer: It creates a scatter plot with the column 1 and column 2 of the dataset.

```
plt.show() #[8]
```

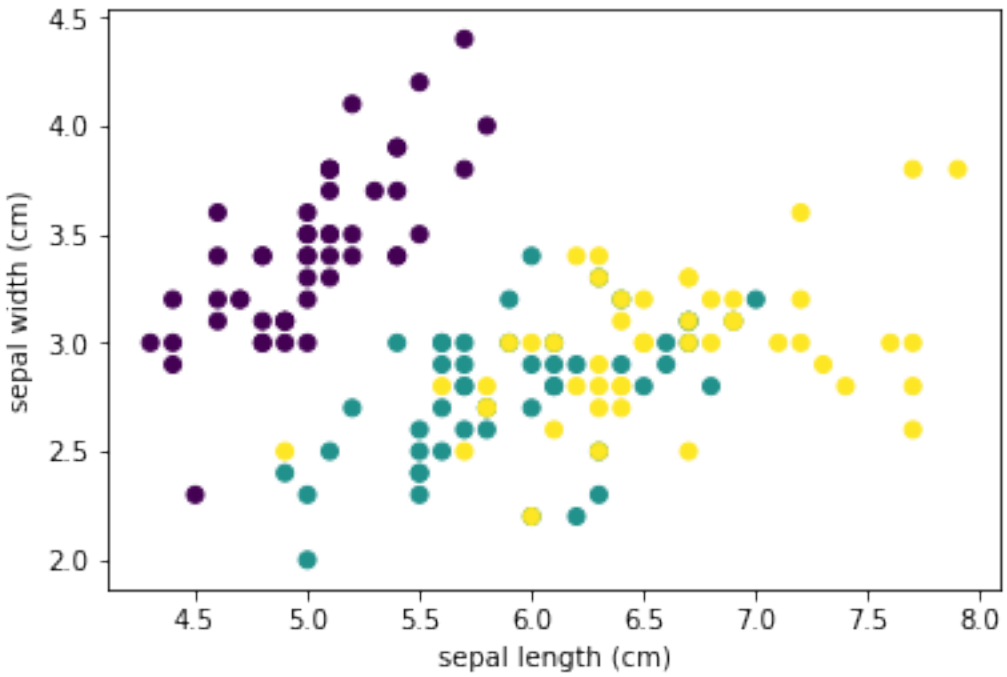
Output:



Answer: The plt.show() command outputs the plot created, which displays the 3 classes of the Iris plant.

```
plt.xlabel(iris_dataset.feature_names[x])
plt.ylabel(iris_dataset.feature_names[y]) # [9]
plt.scatter(X[:, x], X[:, y], c=Y)
plt.show()
```

Output:



Answer: It adds the labels to the plots which displays the 3 classes of the Iris plant.

```
print(Y==0) #[10]
```

Output:

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False]
```

Answer: It takes the first target variable and compare it's value with every target value in each row, if it is same it returns true and if its not than it returns false.

```
print(X[Y==0]) #[11]
```

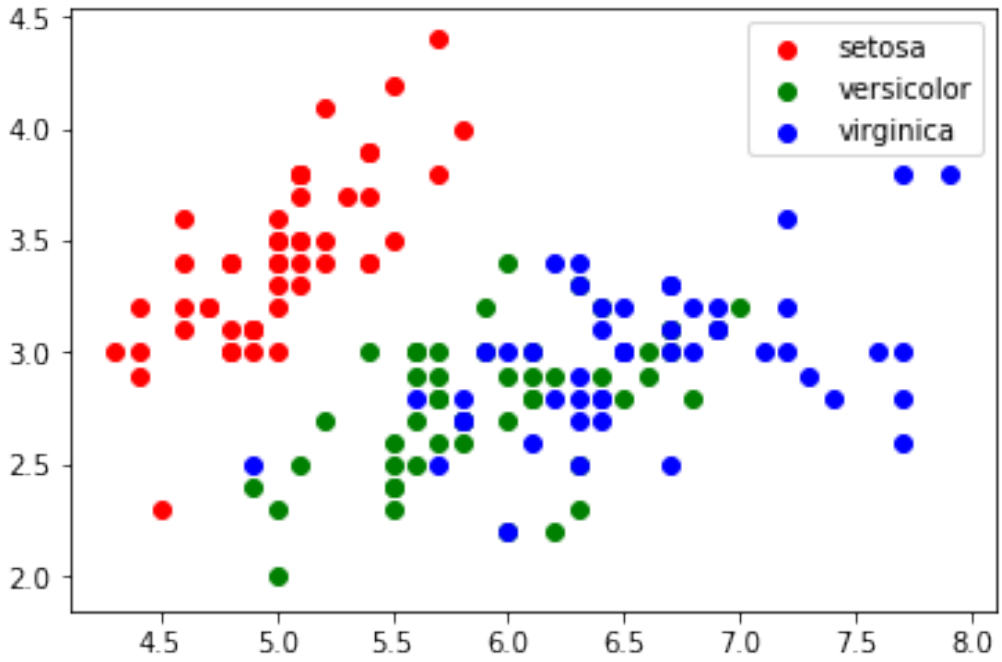
Answer: It returned all the features for which the target value was equal to 0 i.e. Setosa.

```
print(X[Y==0][:, x])
plt.scatter(X[Y==0][:, x], X[Y==0][:, y],          #[12]
            c="red", label=irisData.target_names[0])
plt.scatter(X[Y==1][:, x], X[Y==1][:, y],
            c="green", label=irisData.target_names[1])
plt.scatter(X[Y==2][:, x], X[Y==2][:, y],
            c="blue", label=irisData.target_names[2])
```

Answer: The first scatter plot shows all the values of column 1 for which the target value is equal to Setosa on X axis and all the values of column 2 for which the target value is equal to Setosa on Y axis. Same analogy is used for the target value Virginica and Versicolor.

```
plt.legend()      #[13]
plt.show()
```

Output:



Answer: The legend adds a legend to the scatter plot for the corresponding target values.

```
#pract1prog1.py
import sys
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold
from sklearn import neighbors
import matplotlib.pyplot as plt
irisData = load_iris()
X = irisData.data
```

```
Y = irisData.target
x = 0
y = 1
X = irisData.data
Y = irisData.target
kf = KFold(n_splits=10, shuffle=True)
scores = []
for k in range(1,30):
        score = 0
        clf = neighbors.KNeighborsClassifier(k)
        for learn,test in kf.split(X):
                X_train = X[learn]
                Y_train = Y[learn]
                clf.fit(X_train, Y_train)
                X_test = X[test]
                Y_test = Y[test]
                score = score + clf.score(X_test, Y_test)
        scores.append(score)
print(scores)
print("best k:", scores.index(max(scores))+1)
```

3 Exercise 2: Create your own dataset

In this exercise we created our own dataset by using the online tool from `https://www.librec.net/datagen.html`. The contents of the dataset are as follows,

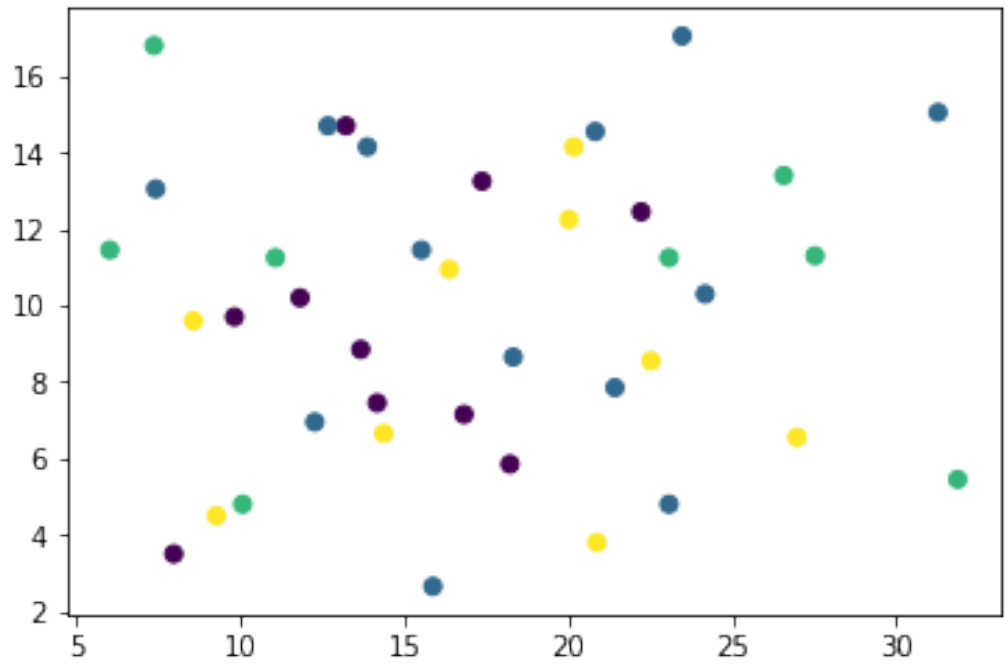
```
from numpy import genfromtxt
my_data = genfromtxt('my_data_gen.csv', delimiter=',') #data load
print(my_data) #prints the contents of the database
print("Length=",len(my_data)) #prints the length of the database
```

Output:

```
[[ 11.825  10.2     1.   ]
 [  9.825   9.7     1.   ]
 [ 13.225  14.7     1.   ]
 [ 16.825   7.15    1.   ]
 [ 22.225  12.45    1.   ]
 [ 14.175   7.45    1.   ]
 [  7.975   3.5     1.   ]
 [ 13.675   8.85    1.   ]
 [ 17.375  13.25    1.   ]
 [ 18.225   5.85    1.   ]
 [ 13.875  14.15    2.   ]
 [ 18.325   8.65    2.   ]
 [ 20.825  14.55    2.   ]
 [ 23.475  17.05    2.   ]
 [ 21.425   7.85    2.   ]
 [ 12.275   6.95    2.   ]
 [  7.425  13.05    2.   ]
 [ 12.675  14.7     2.   ]
 [ 15.525  11.45    2.   ]
 [ 24.175  10.3     2.   ]
 [ 31.275  15.05    2.   ]
 [ 23.075   4.8     2.   ]
 [ 15.875   2.65    2.   ]
 [  7.375  16.8     3.   ]
 [ 11.075  11.25    3.   ]
 [  6.025  11.45    3.   ]
 [ 10.075   4.8     3.   ]
 [ 23.075  11.25    3.   ]
 [ 26.575  13.4     3.   ]
 [ 31.875   5.45    3.   ]
 [ 27.525  11.3     3.   ]
 [ 16.375  10.95    4.   ]
 [ 14.375   6.65    4.   ]
 [  9.275   4.5     4.   ]
 [  8.575   9.6     4.   ]
 [ 20.175  14.15    4.   ]
 [ 20.875   3.8     4.   ]
 [ 26.975   6.55    4.   ]
 [ 22.525   8.55    4.   ]
 [ 20.025  12.25    4.   ]]
Length= 40
```

```
from matplotlib import pyplot as plt
plt.scatter(my_data[:,0], my_data[:,1], c= my_data[:,2])
plt.show()
```

Output:



4 Exercise 3: KNN classifier on IRIS

The KNN algorithm is implemented in the neighbors package. sklearn.neighbors provides functionality for unsupervised and supervised neighbors-based learning methods.

```
from sklearn import neighbors
nb_neighb = 15
help(neighbors.KNeighborsClassifier)
clf = neighbors.KNeighborsClassifier(nb_neighb)

clf.fit(X, Y)    #[1]
```

Output: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=15, p=2, weights='uniform')

Answer: It fits the feature variables with the target variables and learn a model, but for the KNN there is no hypothesis learned because KNN is a lazy algorithm which make local decision by computing distances.

```
print(clf.predict([[ 5.4, 3.2, 1.6, 0.4]]))      #[2]
```

Output: [0]

Answer: The given data features predicts it to belong to the class 0 i.e. Setosa.

```
print(clf.predict_proba([[ 5.4, 3.2, 1.6, 0.4]])) #[3]
```

Output: [[1. 0. 0.]]

Answer: The probability for the given example belong to class Setosa is 1.

```
print(clf.score(X,Y)) #[4]
```

Output: 0.986666666667

Answer: The accuracy of the model is 0.98667.

```
Z = clf.predict(X) #[5]
print(Z)
```

Output: [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 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 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 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]

Answer: It returns the predicted value for all the 150 examples in Iris dataset.

```
print(X[Z!=Y]) #[6]
```

Output: [[6. 2.7 5.1 1.6] [4.9 2.5 4.5 1.7]]

Answer: It returns those examples for which the target value was predicted wrong.

```
from sklearn.model_selection import train_test_split
import random

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
test_size=0.3,random_state=random.seed())        #[7]
```

Answer: It divides the data into 70% for a training sample and 30% for the test sample.

```
print(X_train.shape)      #[8]
print(X_test.shape)
```

Output:
(105, 4)
(45, 4)

Answer: It prints the dimensions of the training set and the test set.

```
print(X_train[Y_train==0].shape)          #[9]
print(X_train[Y_train==1].shape)
print(X_train[Y_train==2].shape)
```

Output:
(34, 4)
(34, 4)
(37, 4)

Answer:It returns the shape for the each target variable of the dataset in training sample.

```
clf = clf.fit(X_train, Y_train)
Y_pred =clf.predict(X_test)
from sklearn.metrics import confusion_matrix      #[10]
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

Output:

```
[[16  0  0]
 [ 0 14  2]
 [ 0  0 13]]
```

Answer:The sklearn has a functionality of confusion matrix which accepts the true target labels and predicted target lables by the KNN.

[11] Explain in a few lines what you can see (in general) in a confusion matrix.

Answer: The confusion matrix returns a square matrix, which contains in the diagonal elements the number of examples truly classified with correct labels. The other elements in the matrix show falsely classified examples.

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle=True)
for learn,test in kf.split(X):
    print("app : ", learn, " test ", test)

kf = KFold(n_splits=3, shuffle=False)              #[12]
for learn,test in kf.split(X):
    print("app : ", learn, " test ", test)
```

Answer: When the value of shuffle is false than the selected examples are not shuffled and kfold for the cross-validation are selected consecutively.

[13] What happen if you replace the line kf=KFold(n_splits=10, shuffle=True) by kf=KFold(n_splits=3, shuffle=False)

```
##practiprog2.py
import statistics
import sys
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold
from sklearn import neighbors
import matplotlib.pyplot as plt
irisData = load_iris()
X = irisData.data
Y = irisData.target
x = 0
y = 1
X = irisData.data
Y = irisData.target
kf = KFold(n_splits=10, shuffle=True)
#for learn,test in kf.split(X):
   # print("app : ", learn, " test ", test)
scores = []
for k in range(1,30):
```

```
        score = 0
        clf = neighbors.KNeighborsClassifier(k)
        for learn,test in kf.split(X):
                X_train = X[learn]
                Y_train = Y[learn]
                clf.fit(X_train, Y_train)
                X_test = X[test]
                Y_test = Y[test]
                score = score + clf.score(X_test, Y_test)
        scores.append(score)
print(scores)
print("best k:", scores.index(max(scores))+1)
```

Output: [9.6000000000000014, 9.4666666666666668, 9.6666666666666679, 9.5333333333333332, 9.6666666666666679, 9.6000000000000014, 9.6666666666666679, 9.7333333333333343, 9.7333333333333343, 9.6666666666666679, 9.7333333333333343, 9.6666666666666679, 9.6666666666666661, 9.6666666666666679, 9.7333333333333343, 9.6666666666666679, 9.8000000000000007, 9.6666666666666679, 9.7333333333333343, 9.6666666666666679, 9.6666666666666679, 9.5333333333333332, 9.6666666666666679, 9.4666666666666686, 9.533333333333335, 9.533333333333335, 9.4000000000000004, 9.4666666666666686, 9.4666666666666686]
best k: 17

Answer: When shuffle = False the value of k is constant at 1 When shuffle = True, the chances of getting better k scores are high in this case almost 10.0

5 Exercise 4: Decision Trees on IRIS

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

```
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()

clf = tree.DecisionTreeClassifier()      #[1]
```

Answer: The clf gets the classifier of Decision Tree.

```
clf = clf.fit(iris.data, iris.target)    #[2]
```

Answer:It learns the tree with the data and the target variables.

```
print(clf.predict([iris.data[50,:]]))    #[3]
```

Output: [1]

Answer: The 50th example is predicted with a label class 1.

```
print(clf.score(iris.data, iris.target))          #[4]
```

Output: 1.0

Answer: The decision tree gives an score of 1.0 . So, it learns perfectly.

```
tree.export_graphviz(clf, out_file='tree.dot')  #[5]
```

Answer: It exports the tree that is being learned by the Decision Tree.

[6] Write a program pract1prog3.py which opens the IRIS dataset and trains a decision tree using the default parameters. Visualize this tree. How many leaves does it contain? Start the training phase again by gradually decreasing the number of leaves from 9 to 3 using the command clf = tree.DecisionTreeClassifier(max_leaf_nodes=xx) and observe (and describe) the resulting trees.

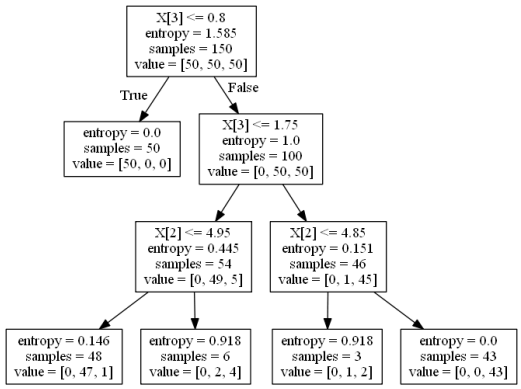
```
#pract1prog3.py

from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
clf = tree.DecisionTreeClassifier(max_leaf_nodes= 3)
clf = clf.fit(iris.data, iris.target)

tree.export_graphviz(clf, out_file='tree1.dot')
```

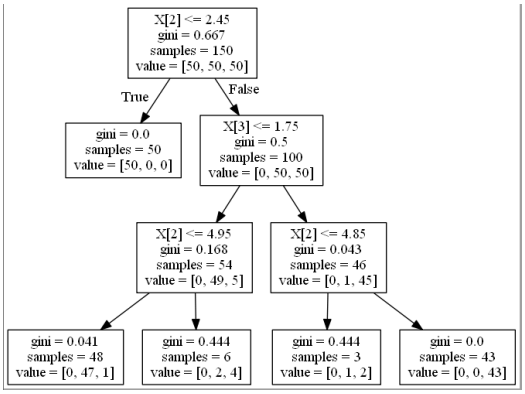
Answer: It has got 7 leaves. When we set the max_leaf_nodes equal to 3, the size of the nodes gets reduced.

[7] Write a program pract1prog4.py which opens the IRIS dataset and trains a decision tree using the Gini criterion (in gini-iris.dot) and a second one using the entropy (in entropy-iris.dot). Compare the two trees.

```
#pract1prog4.py

clf = tree.DecisionTreeClassifier(criterion='gini')
clf = clf.fit(iris.data, iris.target)
tree.export_graphviz(clf, out_file='gini-iris.dot')


clf = tree.DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(iris.data, iris.target)
tree.export_graphviz(clf, out_file='entropy-iris.dot')
```

Output:





Answer: The trees produced using the 'Gini' and the 'Entropy' criterion are exactly the same with the only difference being the criteria used to decide which attribute to split on.

[8] Write a program pract1prog5.py which creates a dataset using the command X,Y = make_classification(n_samples=100000,n_features=20, n_informative=15,n_classes=3), split the generated data into a learning set and a test set (30% for test). Learn a decision tree on the learning set using the command clf = tree.DecisionTreeClassifier(max_leaf_nodes=500*i) (where "i" varies from 1 to 20) then print the score of the classifier on the learning AND on the test set. What do you notice? What is the name of the observed phenomenon?

```
#pract1prog5.py

from sklearn.model_selection import train_test_split
import random
from sklearn import datasets
from sklearn import tree
def main():
    #generate random dataset X, Y
    X, Y = datasets.make_classification(n_samples=100000,
n_features=20,n_informative=15,n_classes=3)
    #split data into train and test data
    X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.3,
random_state=random.seed())
    #iterate i 20times
    for i in range(1,20):
        #set a score value
        x = 0
        y = 0
        #call our classification function
        clf = tree.DecisionTreeClassifier(max_leaf_nodes=500*i)
        #fit X_train and Y_train to decision tress classifier
        clf.fit(X_train, Y_train)
        #after calculating the score for both the train and test
        #We noticed the phonenon called: Overfitting
        y = y + clf.score(X_train, Y_train)
        x = x + clf.score(X_test, Y_test)
        print("Test: %6.4f" %x,"Train: %6.4f" %y)


if __name__ == '__main__':
    main()
```

Output:
Test: 0.8111 Train: 0.8515
Test: 0.8288 Train: 0.8892

Test: 0.8323 Train: 0.9127
Test: 0.8334 Train: 0.9300
Test: 0.8334 Train: 0.9440
Test: 0.8319 Train: 0.9541
Test: 0.8282 Train: 0.9617
Test: 0.8253 Train: 0.9691
Test: 0.8280 Train: 0.9765
Test: 0.8278 Train: 0.9839
Test: 0.8245 Train: 0.9889
Test: 0.8205 Train: 0.9935
Test: 0.8188 Train: 0.9977
Test: 0.8150 Train: 1.0000
Test: 0.8157 Train: 1.0000
Test: 0.8169 Train: 1.0000
Test: 0.8149 Train: 1.0000
Test: 0.8159 Train: 1.0000
Test: 0.8148 Train: 1.0000

Answer: After calculating the score for both the train and test, we noticed the phenomenon called: Overfitting.

[9] Same question (pract1prog6.py) but with trees of different depth using the command clf = tree.DecisionTreeClassifier(max_depth=i) (for i from 1 to 40).

```
#pract1prog6.py


def main():
    #generate random dataset X, Y
    X, Y = datasets.make_classification(n_samples=100000,n_features=20,
n_informative=15,n_classes=3)
    #split data into train and test data
    X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
test_size = 0.3, random_state=random.seed())
    #iterate i 20times
    for i in range(1,40):
        #set a score value
        x = 0
```

```
        y = 0
        #i = [i+1 for i in range(40)]
        #call our classification function
        clf = tree.DecisionTreeClassifier(max_leaf_nodes=500*i,
max_depth = i+1)
        #fit X-train and Y-train to decision tress classifier
        clf.fit(X_train, Y_train)
        #after calculating the score for both the train and test
        #We noticed the phonenon called: Overfitting
        y = y + clf.score(X_train, Y_train)
        x = x + clf.score(X_test, Y_test)
        print("Test: %6.4f" %x,"Train: %6.4f" %y)


if __name__ == '__main__':
    main()
```

Output:

```
Test: 0.5957 Train: 0.6009
Test: 0.6315 Train: 0.6327
Test: 0.6562 Train: 0.6637
Test: 0.6903 Train: 0.7037
Test: 0.7135 Train: 0.7325
Test: 0.7395 Train: 0.7649
Test: 0.7668 Train: 0.8025
Test: 0.7852 Train: 0.8362
Test: 0.7982 Train: 0.8674
Test: 0.8034 Train: 0.8979
Test: 0.8070 Train: 0.9238
Test: 0.8087 Train: 0.9447
Test: 0.8061 Train: 0.9611
Test: 0.8054 Train: 0.9728
Test: 0.8024 Train: 0.9816
Test: 0.8015 Train: 0.9880
Test: 0.8001 Train: 0.9915
Test: 0.7985 Train: 0.9941
Test: 0.7996 Train: 0.9960
Test: 0.7993 Train: 0.9973
Test: 0.7946 Train: 0.9983
Test: 0.7974 Train: 0.9988
Test: 0.7955 Train: 0.9994
```

Test: 0.7964 Train: 0.9997
Test: 0.7964 Train: 0.9998
Test: 0.7956 Train: 0.9999
Test: 0.7958 Train: 1.0000
Test: 0.7949 Train: 1.0000
Test: 0.7953 Train: 1.0000
Test: 0.7967 Train: 1.0000
Test: 0.7973 Train: 1.0000
Test: 0.7965 Train: 1.0000
Test: 0.7962 Train: 1.0000
Test: 0.7954 Train: 1.0000
Test: 0.7975 Train: 1.0000
Test: 0.7972 Train: 1.0000
Test: 0.7953 Train: 1.0000
Test: 0.7977 Train: 1.0000
Test: 0.7968 Train: 1.0000

Answer: After calculating the score for both the train and test, we noticed the same Overfitting phenomenon.

6 Exercise 5: Neural Networks on DIGITS

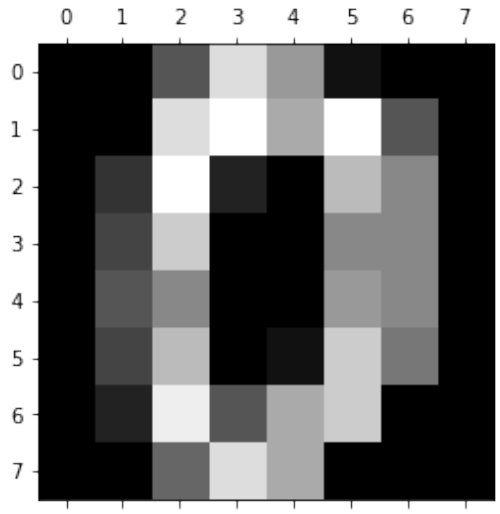
The digits dataset contains 5620 instances that are described by 64 attributes (corresponding to the 8*8 images of integer pixels in the range 0(white)..16(black). The target is a digit between 0 and 9. The MNIST dataset also contains digits (but they are 28*28 images and are more numerous).

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import random
digits = load_digits()
digits.data[0]
digits.images[0]
digits.data[0].reshape(8,8)
digits.target[0]
```

Output: 0

```
from matplotlib import pyplot as plt
plt.matshow(digits.images[0])
plt.gray()
plt.show()
```

Output:



```
Y=digits.target
print(len(Y[Y==0]))
```

Output: 178

```
from sklearn.neural_network import MLPClassifier
X = digits.data
```

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(5, 2),random_state=1)                    #[1]
clf.fit(X, Y)
```

Output:
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(5, 2), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

Answer: This command is used to tune the parameters of the multi-layered perceptron with two hidden layer with 5 and 2 neurons in every layer respectively.

[2] Write a program pract1prog7.py which opens the DIGITS dataset and train a MLP using the default parameters and a small number of hidden layers. Split the data into a learning set and a test set (30% for test) and report the score obtained by your classifier on the test set. Run your program multiple times with the same parameters. Change at least 3 parameters of your classifier (ex: number of neurons, number of layers, learning rate) and report the score your obtain on your test set for each version. What can you conclude?

```
#pract1prog7.py

#The MLP is made on digits dataset while tuning different parameters
given below.

from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import random
digits = load_digits()
digits.data[0]
digits.images[0]
digits.data[0].reshape(8,8)
digits.target[0]

TaX= digits.data
TX= digits.target
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(20,10, 10,8,9),random_state=1)
X_train,X_test,Y_train,Y_test = train_test_split(TaX,TX,
test_size=0.3,random_state=random.seed())
print(X_train.shape)
print(X_test.shape)

clf.fit(X_train, Y_train)

Y_pred =clf.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
print(clf.score(X_test,Y_test))

clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
```

```
hidden_layer_sizes=(10,10,10, 12),random_state=1,
learning_rate= 'adaptive')
clf.fit(X_train, Y_train)
Y_pred =clf.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
print(clf.score(X_test,Y_test))
```

Output:

```
[[50  0  0  0  1  0  0  0  0  0]
 [ 0 42  0  0  0  0  1  0  2  1]
 [ 2  1 43  0  0  9  1  0  0  0]
 [ 0  0  0 50  0  6  0  2  2  3]
 [ 0  0  0  0 44  0  1  0  1  1]
 [ 0  0  1  3  1 43  0  0  4  0]
 [ 0  0  0  0  1  0 58  0  0  0]
 [ 1  0  0  2  0  0  0 46  1  2]
 [ 0  2  1  1  0  1  0  0 47  5]
 [ 0  1  1  5  1  0  0  1  5 43]]
0.862962962963
```

```
[[48  0  0  0  0  0  2  0  0  1]
 [ 0 44  0  0  0  0  0  0  2  0]
 [ 0  0 54  2  0  0  0  0  0  0]
 [ 0  0  0 56  0  0  0  0  2  5]
 [ 0  0  0  0 45  0  0  2  0  0]
 [ 1  1  0  0  0 46  1  0  1  2]
 [ 0  0  0  0  0  0 58  0  1  0]
 [ 0  2  0  1  1  0  0 48  0  0]
 [ 0  2  0  0  0  1  1  0 48  5]
 [ 0  0  0  5  0  2  0  1  6 43]]
0.907407407407
```

Answer: Within the program the learning rate layer and alpha was changed and it was sen that the effect tunning the activation parameter was significant on the training and test data and increasing the layer influenced the MLP to learn more complex model and perform better.

7 Exercise 6: Play with your own data

[1] Write a program pract1prog8.py which open your dataset, train a KNN classifier (with k= 3) on the data and print the score on the training set.

```
from sklearn import neighbors
import numpy as np
import pandas as pd
nb_neighb = 3
clf = neighbors.KNeighborsClassifier(nb_neighb)


a= ['Feature 1', 'Feature 2', 'Target']
data= pd.read_csv('my_data_gen.csv', names= a)
data.head()
```

Output:

	Feature 1	Feature 2	Target
0	11.825	10.20	1
1	9.825	9.70	1
2	13.225	14.70	1
3	16.825	7.15	1
4	22.225	12.45	1

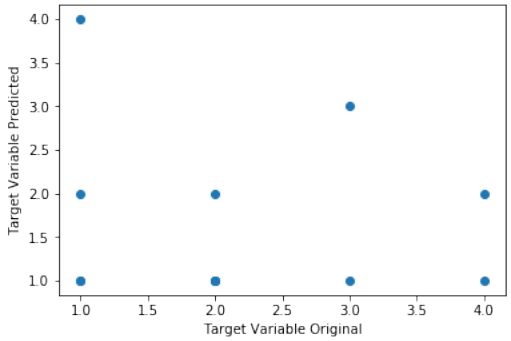
```
features_data= data.iloc[:,0:2]
target_data= np.ravel(data.iloc[:,2:3])
target_data1= data.iloc[:,2:3]
clf.fit(features_data,target_data)
clf.score(features_data,target_data)
```

Output: 0.55000000000000004
Answer: The score of the KNN is 0.5500

[2] Extend the previous program to split your dataset into training and test (30% for test), train a classifier using a KNN algorithm (with k = 3) on the training set and evaluate it on the test set. Print a number of examples that are not well classified.

```
from sklearn.model_selection import train_test_split
import random

X_train,X_test,Y_train,Y_test = train_test_split(features_data,target_data,
test_size=0.3,random_state=random.seed())
clf.fit(X_train,Y_train)
Y_pred=clf.predict(X_test)
print("Target Variable Predicted",Y_pred )
print ("Target Variable Original", Y_test)
for i in range(1,12):
    if Y_pred[i]==Y_test[i]:
        print('The correctly predicted examples are', i)
    else:
        print('The incorrectly predicted examples are', i)

plt.scatter(Y_test,Y_pred)
plt.xlabel('Target Variable Original')
plt.ylabel('Target Variable Predicted')
plt.show()
```

Output:
Target Variable Predicted [2 1 1 1 2 4 1 1 2 1 1 3]
Target Variable Original [1 4 2 3 2 1 2 2 4 1 1 3]
The incorrectly predicted examples are 1
The incorrectly predicted examples are 2
The incorrectly predicted examples are 3
The correctly predicted examples are 4
The incorrectly predicted examples are 5
The incorrectly predicted examples are 6
The incorrectly predicted examples are 7
The incorrectly predicted examples are 8
The correctly predicted examples are 9
The correctly predicted examples are 10
The correctly predicted examples are 11



Answer: The number of examples predicted correctly are low with k=3

[3]Extend the previous program to evaluate its accuracy using a leave-one-out cross validation.

```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
avg_score =[]
for train_index, test_index in loo.split(X):

        X_trian, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        clf.fit(X_trian, Y_train)
        avg_score.append(clf.score(X_test,Y_test))

print("\nThe average score using LeaveOneOut Cross-validation : " +
        str(sum(avg_score)/len(avg_score)))

print("The scores over 25 iterations are : \n" + str(avg_score))
```

```
#pract1prog8.py

from sklearn import neighbors
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import random

#1
nb_neighb = 3
clf = neighbors.KNeighborsClassifier(nb_neighb)

a= ['Feature 1', 'Feature 2', 'Target']
data= pd.read_csv('my_data_gen.csv', names= a)
data.head()

features_data= data.iloc[:,0:2]
target_data= np.ravel(data.iloc[:,2:3])
target_data1= data.iloc[:,2:3]
clf.fit(features_data,target_data)
clf.score(features_data,target_data)
#2
X_train,X_test,Y_train,Y_test = train_test_split(features_data,target_data,
test_size=0.3,random_state=random.seed())
clf.fit(X_train,Y_train)
Y_pred=clf.predict(X_test)
print("Target Variable Predicted",Y_pred )
print ("Target Variable Original", Y_test)
for i in range(1,12):
    if Y_pred[i]==Y_test[i]:
        print('The correctly predicted examples are', i)
    else:
        print('The incorrectly predicted examples are', i)

plt.scatter(Y_test,Y_pred)
plt.xlabel('Target Variable Original')
plt.ylabel('Target Variable Predicted')
plt.show()
#3
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
avg_score =[]
for train_index, test_index in loo.split(X):

        X_trian, X_test = X[train_index], X[test_index]
```

```
        Y_train, Y_test = Y[train_index], Y[test_index]
        clf.fit(X_trian, Y_train)
        avg_score.append(clf.score(X_test,Y_test))

print("\nThe average score using LeaveOneOut Cross-validation : " +
        str(sum(avg_score)/len(avg_score)))

print("The scores over 25 iterations are : \n" + str(avg_score))
```

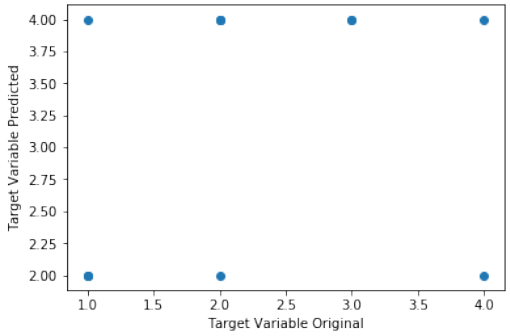
[4] Do the same process as in the previous three questions in a program pract1prog9.py with a Decision tree classifier (report the parameters that you are using).

```
from sklearn import tree
clf1= tree.DecisionTreeClassifier(criterion= 'entropy', max_leaf_nodes=20)
clf1.fit(features_data,target_data)
clf1.score(features_data,target_data)
```

Output: 0.875

```
clf1.fit(X_train,Y_train)
Y_pred=clf1.predict(X_test)
print("Target Variable Predicted",Y_pred )
print ("Target Variable Original", Y_test)
for i in range(1,12):
    if Y_pred[i]==Y_test[i]:
        print('The correctly predicted examples are', i)
    else:
        print('The incorrectly predicted examples are', i)

plt.scatter(Y_test,Y_pred)
plt.xlabel('Target Variable Original')
plt.ylabel('Target Variable Predicted')
plt.show()
```

Output:
Target Variable Predicted [2 2 4 4 2 2 4 4 4 2 4 4]
Target Variable Original [1 4 2 3 2 1 2 2 4 1 1 3]
The incorrectly predicted examples are 1
The incorrectly predicted examples are 2
The incorrectly predicted examples are 3
The correctly predicted examples are 4
The incorrectly predicted examples are 5
The incorrectly predicted examples are 6
The incorrectly predicted examples are 7
The correctly predicted examples are 8
The incorrectly predicted examples are 9
The incorrectly predicted examples are 10
The incorrectly predicted examples are 11



```
from sklearn.tree import export_graphviz
loo = LeaveOneOut()
avg_score =[]
i = 1
for train_index, test_index in loo.split(X):


        X_trian, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        clf1.fit(X_trian, Y_train)
        avg_score.append(clf1.score(X_test,Y_test))
        tree.export_graphviz(clf1, out_file= 'part_6_3(' +str(i)+')_dt.dot')
        i = i + 1

print("\nThe average score using LeaveOneOut Cross-validation : " +
        str(sum(avg_score)/len(avg_score)))
print("The scores over 25 iterations are : \n" + str(avg_score))
```

```
#pract1prog9.py
from sklearn import tree
clf1= tree.DecisionTreeClassifier(criterion= 'entropy', max_leaf_nodes=20)
clf1.fit(features_data,target_data)
clf1.score(features_data,target_data)

clf1.fit(X_train,Y_train)
Y_pred=clf1.predict(X_test)
print("Target Variable Predicted",Y_pred )
print ("Target Variable Original", Y_test)
for i in range(1,12):
    if Y_pred[i]==Y_test[i]:
        print('The correctly predicted examples are', i)
    else:
```

```
        print('The incorrectly predicted examples are', i)

plt.scatter(Y_test,Y_pred)
plt.xlabel('Target Variable Original')
plt.ylabel('Target Variable Predicted')
plt.show()

from sklearn.tree import export_graphviz
loo = LeaveOneOut()
avg_score =[]
i = 1
for train_index, test_index in loo.split(X):


        X_trian, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        clf1.fit(X_trian, Y_train)
        avg_score.append(clf1.score(X_test,Y_test))
        tree.export_graphviz(clf1, out_file= 'part_6_3(' +str(i)+')_dt.dot')
        i = i + 1

print("\nThe average score using LeaveOneOut Cross-validation : " +
        str(sum(avg_score)/len(avg_score)))
print("The scores over 25 iterations are : \n" + str(avg_score))
```

[5] Do the same process as in the previous three questions in a program pract1prog10.py with a Neural Network classifier.

```
from sklearn.neural_network import MLPClassifier
clf2 = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(20,20,20),random_state=1)
clf2.fit(features_data, target_data)
```

Output:
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(20, 20, 20), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

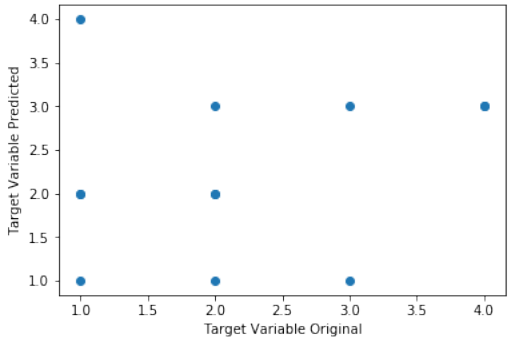
```
clf2.score(features_data,target_data)
```

Output: 0.5

```
clf2.fit(X_train,Y_train)
Y_pred=clf2.predict(X_test)
print("Target Variable Predicted",Y_pred )
print ("Target Variable Original", Y_test)
for i in range(1,12):
    if Y_pred[i]==Y_test[i]:
        print('The correctly predicted examples are', i)
    else:
        print('The incorrectly predicted examples are', i)

plt.scatter(Y_test,Y_pred)
plt.xlabel('Target Variable Original')
plt.ylabel('Target Variable Predicted')
plt.show()
```

Output:
Target Variable Predicted [2 3 2 1 2 2 1 3 3 1 4 3]
Target Variable Original [1 4 2 3 2 1 2 2 4 1 1 3]
The incorrectly predicted examples are 1
The correctly predicted examples are 2
The incorrectly predicted examples are 3

The correctly predicted examples are 4
The incorrectly predicted examples are 5
The incorrectly predicted examples are 6
The incorrectly predicted examples are 7
The incorrectly predicted examples are 8
The correctly predicted examples are 9
The incorrectly predicted examples are 10
The correctly predicted examples are 11




```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
avg_score =[]
for train_index, test_index in loo.split(X):

        X_trian, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        clf.fit(X_trian, Y_train)
        avg_score.append(clf.score(X_test,Y_test))

print("\nThe average score using LeaveOneOut Cross-validation : " +
        str(sum(avg_score)/len(avg_score)))

print("The scores over 25 iterations are : \n" + str(avg_score))
```

```
from itertools import product
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.neural_network import MLPClassifier

 Loading data (Plots will only be created for attributes 'X0' and 'X1')
M,N = make_classification(n_samples = 25,n_features =4,
                          n_redundant = 2, n_classes = 2,
                          n_informative = 2, random_state=10)
M = M[:, [0, 2]]

clfa = DecisionTreeClassifier(max_depth=4, criterion = "entropy")
clfb = KNeighborsClassifier(n_neighbors=5)
clfc = MLPClassifier(hidden_layer_sizes= (10,7,3), learning_rate_init = 0.05,
activation = 'tanh')
# Fitting to the dataset
clfa.fit(M, N)
clfb.fit(M,N)
clfc.fit(M, N)
```

Output:

MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(10, 7, 3), learning_rate='constant', learning_rate_init=0.05, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

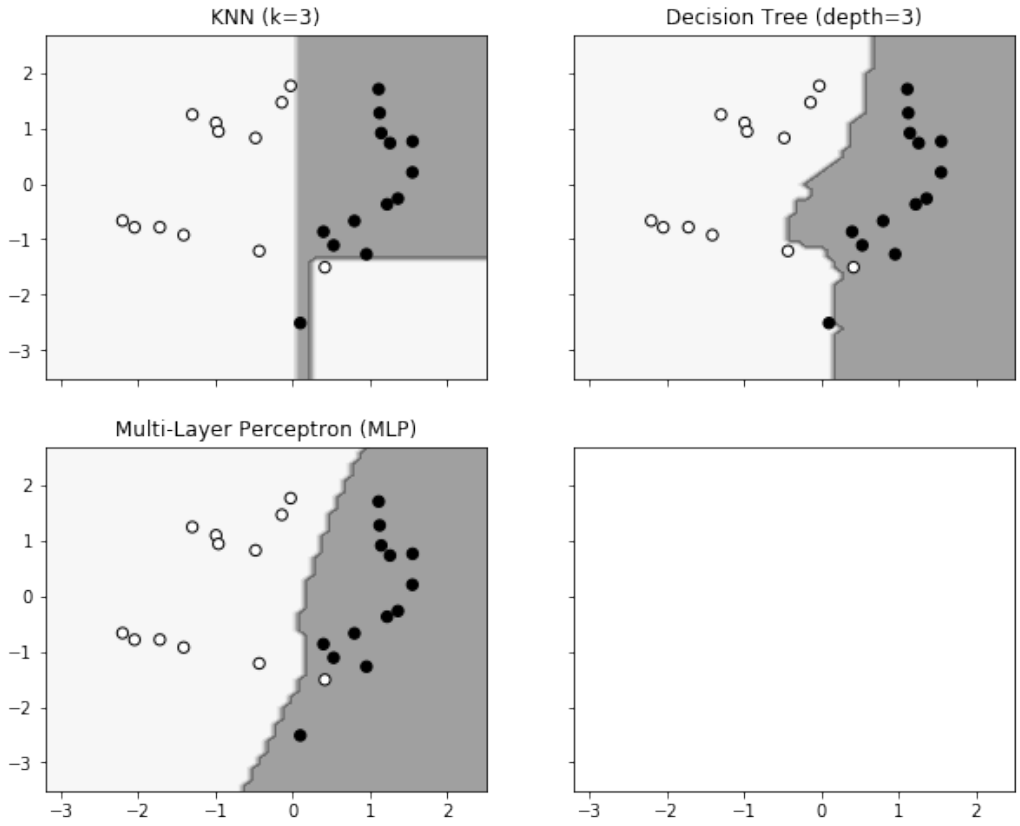
```
#pract1prog10.py

from sklearn.neural_network import MLPClassifier
clf2 = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(20,20,20),random_state=1)
clf2.fit(features_data, target_data)

clf2.score(features_data,target_data)

clf2.fit(X_train,Y_train)
Y_pred=clf2.predict(X_test)
print("Target Variable Predicted",Y_pred )
print ("Target Variable Original", Y_test)
for i in range(1,12):
    if Y_pred[i]==Y_test[i]:
        print('The correctly predicted examples are', i)
    else:
        print('The incorrectly predicted examples are', i)

plt.scatter(Y_test,Y_pred)
plt.xlabel('Target Variable Original')
plt.ylabel('Target Variable Predicted')
plt.show()
#3
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
avg_score =[]
for train_index, test_index in loo.split(X):

        X_trian, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        clf2.fit(X_trian, Y_train)
        avg_score.append(clf2.score(X_test,Y_test))

print("\nThe average score using LeaveOneOut Cross-validation :" +
        str(sum(avg_score)/len(avg_score)))

print("The scores over 25 iterations are :\n" + str(avg_score))
```

[6] Which algorithm can explain your concept best? Print the generated models when it is possible. Do they tell you anything interesting?

Answer: It can be seen from the plots below in output that while KNN seems to slightly overfit the data but both the Decision Tree and MLP seem to generalise the over data better. However, the decision boundary seems to be more robust in the case of MLP.

```
# Plotting decision region

x_min, x_max = M[:, 0].min() - 1, M[:, 0].max() + 1
y_min, y_max = M[:, 1].min() - 1, M[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', figsize=(10, 8))

for idx, clf, tt in zip(product([0, 1], [0, 1]),
                        [clfa, clfb, clfc],
                        ['KNN (k=5)','Decision Tree (depth=4)',
                         'Multi-Layer Perceptron (MLP)']):

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
    axarr[idx[0], idx[1]].scatter(M[:, 0], M[:, 1], c=N, s=40, edgecolor='k')
    axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```

Output:


KNN (k=3)
Decision Tree (depth=3)
Multi-Layer Perceptron (MLP)
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2