

**American University of Sharjah**

**College of Engineering**

**Department of Electrical Engineering**

**Spring 2021**

**CMP 466 Machine Learning & Data Mining**

**Project Assignment 2**

**Farooq Mirza 80205**

**Omar Al Maqoodi 76307**

**Omar Fayed 69702**

1. **Reading the data**

#-----------------------

# Reading the data

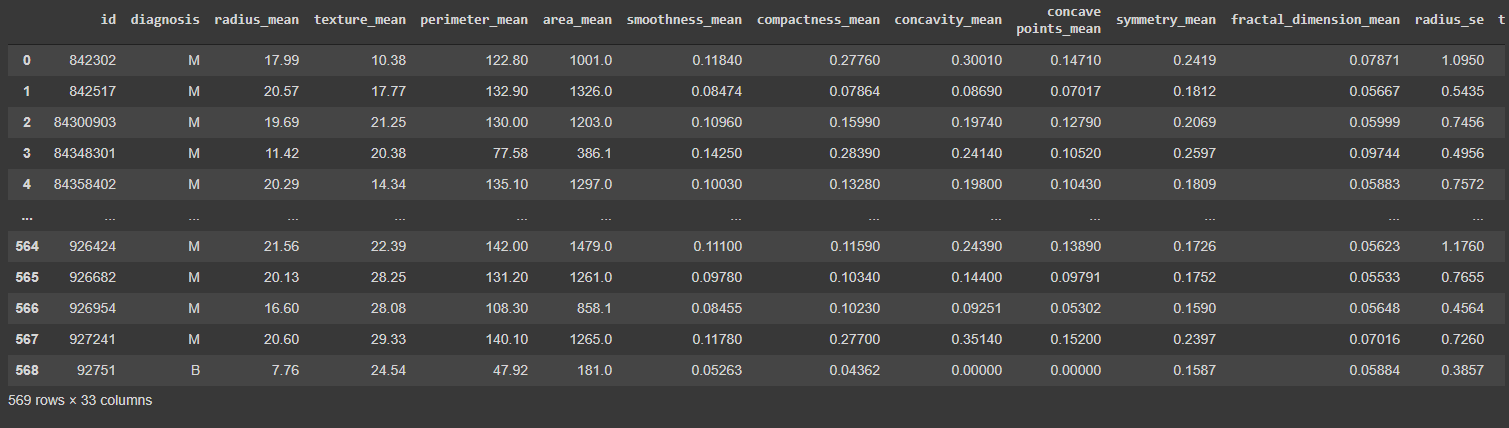
#-----------------------

import pandas as pd

import numpy as np

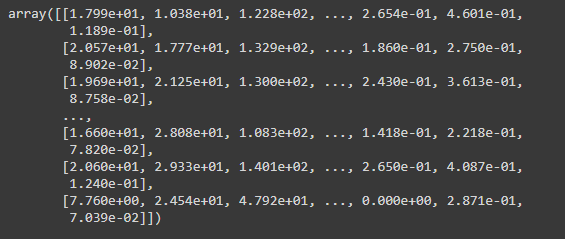
data = pd.read\_csv('Breast Cancer Wisconsin Data Set')

print(data)



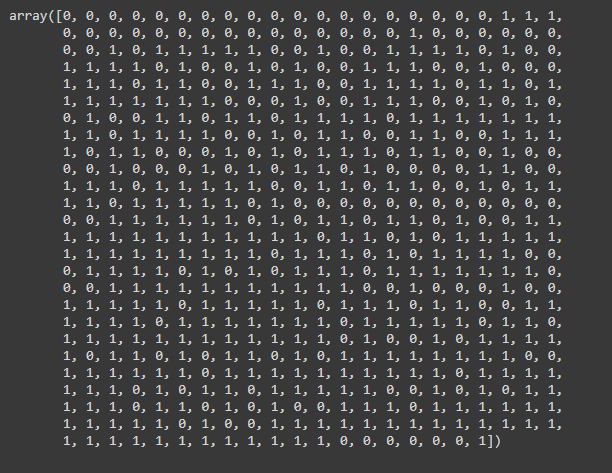
X = data.drop(columns=['id', 'Unnamed: 32','diagnosis']).astype(float).to\_numpy()

print(X)



y = data.replace(['M','B'],[0, 1])['diagnosis'].astype(int).to\_numpy()

print(y)



We read the data from the .csv file as a pandas data frame. Then we got rid of all the unnecessary features and the labels of the dataset while storing it into X, more precisely we got rid of column 32 since it was an Unnamed feature full of Null values. We also convert the dataset type of X to float to ease NumPy’s underlying C calculations. Then we converted the categorical labels to integer ones, which were set as 0 for Malignant which was labeled as ‘M’ and 1 for Benign which was labeled as ‘B’.

1. **Performing Random Splits**

#-----------------------

# Performing random splits

#-----------------------

from sklearn import tree

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

for i in range(0, 15):

    clf = tree.DecisionTreeClassifier()

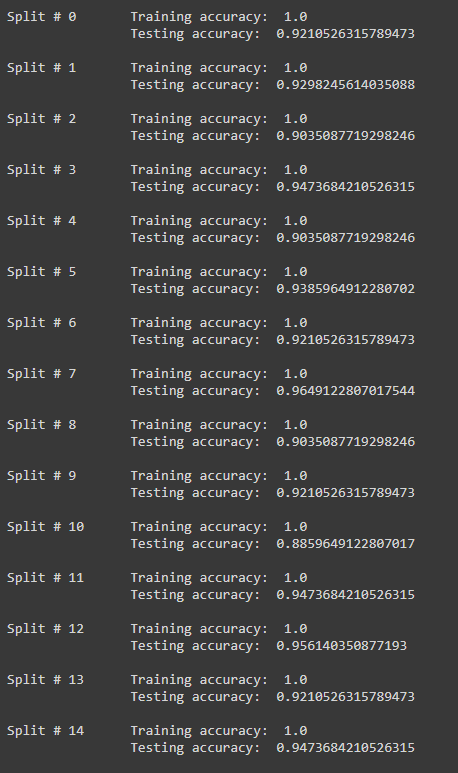
    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

    clf.fit(X\_train, y\_train)

    print("Split #", i, end="\t")

    print("Training accuracy: ", clf.score(X\_train, y\_train))

    print("\t\tTesting accuracy: ", clf.score(X\_test, y\_test), "\n")



We made use of the train\_test\_split() function in order to split our data into splits of 80% Training set and 20% Testing set, by stating that the test size was 0.2 (20%) in the input. We then repeat the split 15 times (0 → 14) and check our results for each separate split. We then infer that the test scores vary between the various different splits.

1. **Performing K-Fold Cross Validation**

#-----------------------

# K-Fold Cross Validation

#-----------------------

from sklearn.model\_selection import cross\_validate

print("k-fold Cross Validation Results")

cv\_results = cross\_validate(clf, X, y, cv=5)

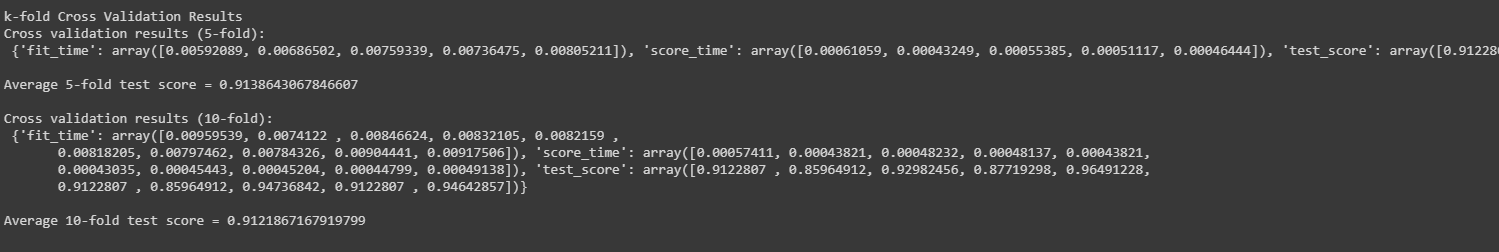
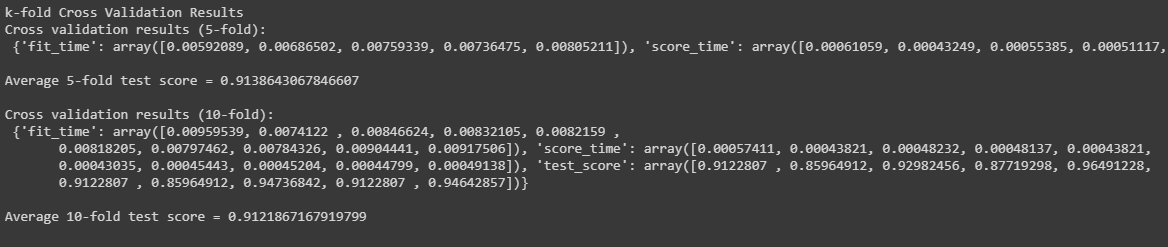
print("Cross validation results (5-fold):\n", cv\_results)

print("\nAverage 5-fold test score =", np.average(cv\_results['test\_score']), "\n")

cv\_results = cross\_validate(clf, X, y, cv=10)

print("Cross validation results (10-fold):\n", cv\_results)

print("\nAverage 10-fold test score =", np.average(cv\_results['test\_score']), "\n")



1. **Tweaking Hyperparameters**
   1. max\_depth

#-----------------------

# Tweaking Hyperparameters

#-----------------------

import matplotlib.pyplot as plt

clf = tree.DecisionTreeClassifier()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

clf.fit(X\_train, y\_train)

tree.plot\_tree(clf)

A picture containing diagram

Description automatically generated

test\_scores = []

nodes = []

for i in range(1, 9):

    clf = tree.DecisionTreeClassifier(max\_depth=i)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

    clf.fit(X\_train, y\_train)

    cv\_results = cross\_validate(clf, X, y, cv=5)

    test\_scores.append(np.average(cv\_results['test\_score']) \* 100)

    nodes.append(clf.tree\_.node\_count)

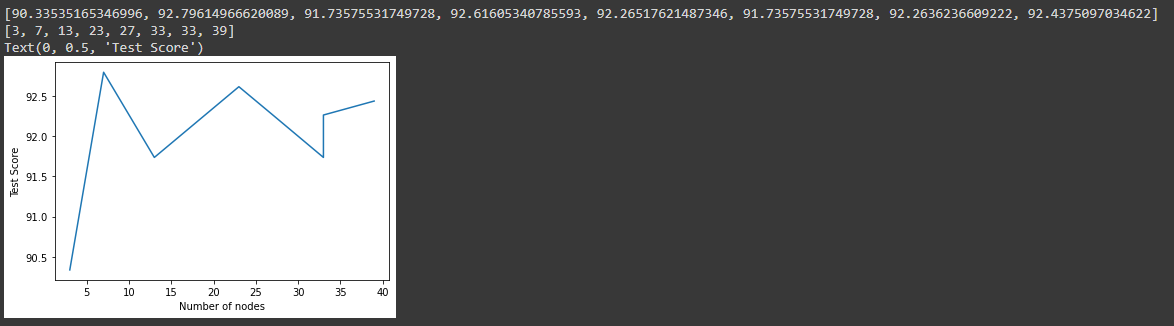
print(test\_scores)

print(nodes)

plt.plot(nodes, test\_scores)

plt.xlabel("Number of nodes")

plt.ylabel("Test Score")

****

We first run the decision tree classifier to check the maximum depth of the decision tree which we obtain to be 8. We then vary the max\_depth hyper parameter of the decision tree classifier between 1 and 8, fit the data, cross validate using k fold 5 and then find the averages in percentage form, which we append to a list. We also contain all the number of nodes using clf.tree\_.node\_count to be able to plot the prior 2 arrays against one another. The results show that a max depth of 4 provides a test score of more than 92.5%.

**b. min\_samples\_split**

test\_scores = []

splits = []

for i in range(1, 8):

    clf = tree.DecisionTreeClassifier(min\_samples\_split=60 \* i)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

    clf.fit(X\_train, y\_train)

    cv\_results = cross\_validate(clf, X, y, cv=5)

    test\_scores.append(np.average(cv\_results['test\_score']) \* 100)

    splits.append(60 \* i)

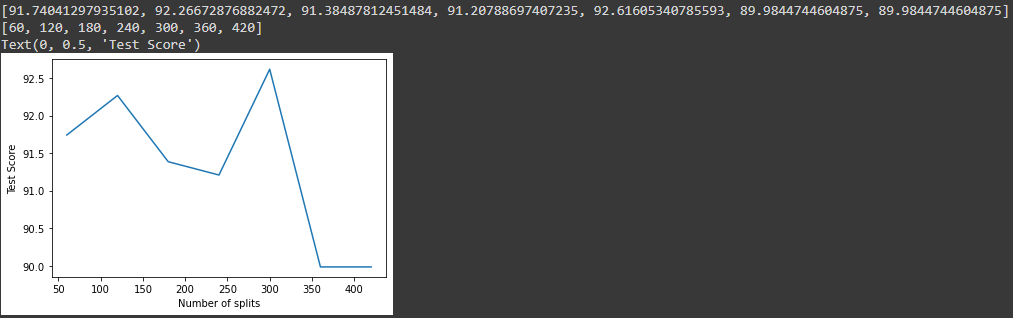
print(test\_scores)

print(splits)

plt.plot(splits, test\_scores)

plt.xlabel("Number of splits")

plt.ylabel("Test Score")



We vary the min\_sample\_splits hyper parameter of the decision tree classifier between 1 x 60 (60) and 8 x 60 (480), we then follow the same steps as in 4(a) by fittng the data, cross validating it using k fold 5 and then find the averages in percentage form, which we append to a list. We also contain all the number of splits in an array to be able to plot the splits and test score arrays against one another.

The fluctuations for both graphs are extremely sharp because of the fact our data is very pure, our largest max depth is 8 and the best one is 4, the test scores for both K=5 and K=10 are very high, more than 90%. The results also show that the best number of splits for our data is 300 splits with a test score of 92.5.

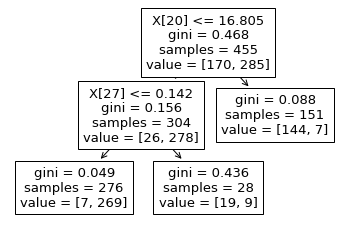
1. **Best fitted tree**

clf = tree.DecisionTreeClassifier(max\_depth=4, min\_samples\_split=300)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

clf.fit(X\_train, y\_train)

tree.plot\_tree(clf)



The tests performed above result in a very simplified Decision Tree with a depth of 4 and a split of 300 for K=5. We believe that due to the fact this data is extremely refined and pure our Decision Tree is very accurate and Optimized.