Lab Course Machine Learning Kalaiselvan Panneerselvam

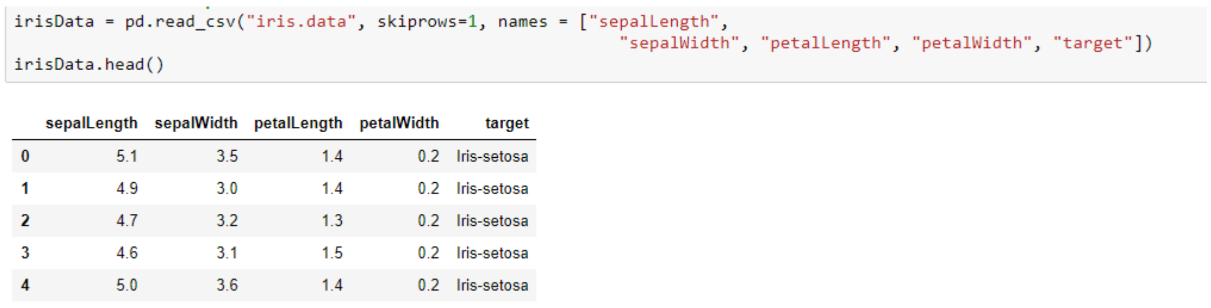
Exercise 7 278920

Exercise 1: Implement K-Nearest Neighbor (KNN) (10 Points)

1 Data Preprocessing

– Load the data using pandas & dropping NA rows

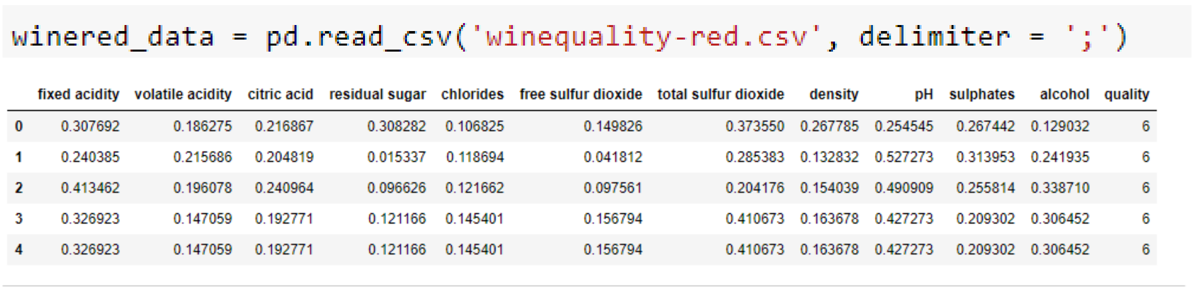
Iris Dataset



```
irisData = pd.read_csv("iris.data", skiprows=1, names = ["sepalLength",
                                                          "sepalWidth", "petalLength", "petalWidth", "target"])
irisData.head()
```

	sepalLength	sepalWidth	petalLength	petalWidth	target
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Wine Dataset

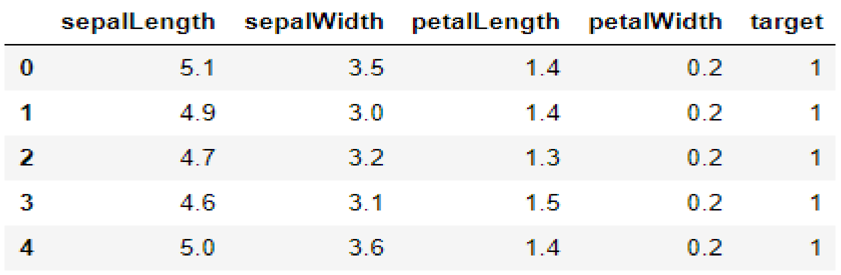


```
winered_data = pd.read_csv('winequality-red.csv', delimiter = ';')
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0.307692	0.186275	0.216867	0.308282	0.106825	0.149826	0.373550	0.267785	0.254545	0.267442	0.129032	6
1	0.240385	0.215686	0.204819	0.015337	0.118694	0.041812	0.285383	0.132832	0.527273	0.313953	0.241935	6
2	0.413462	0.196078	0.240964	0.096626	0.121662	0.097561	0.204176	0.154039	0.490909	0.255814	0.338710	6
3	0.326923	0.147059	0.192771	0.121166	0.145401	0.156794	0.410673	0.163678	0.427273	0.209302	0.306452	6
4	0.326923	0.147059	0.192771	0.121166	0.145401	0.156794	0.410673	0.163678	0.427273	0.209302	0.306452	6

- Convert any non-numeric values to numeric values

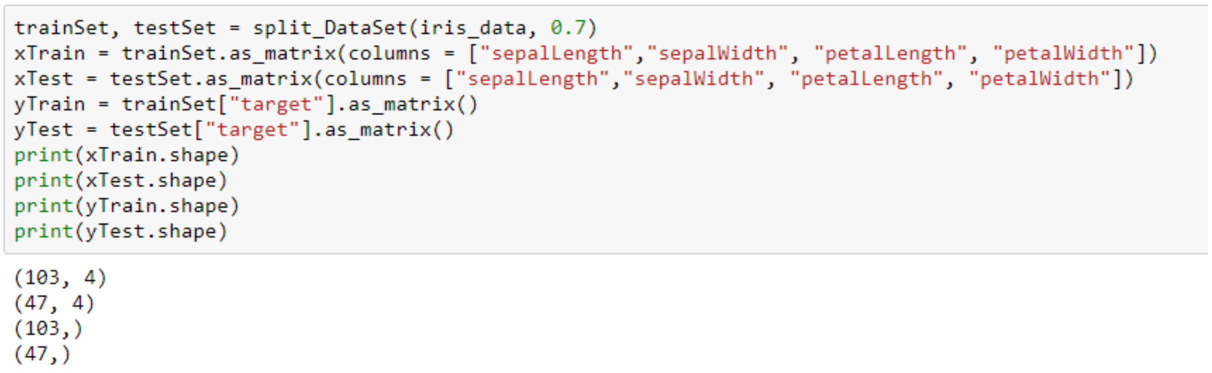
Iris Dataset



	sepalLength	sepalWidth	petalLength	petalWidth	target
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

- Split the data into a train(70%) and test(30%)

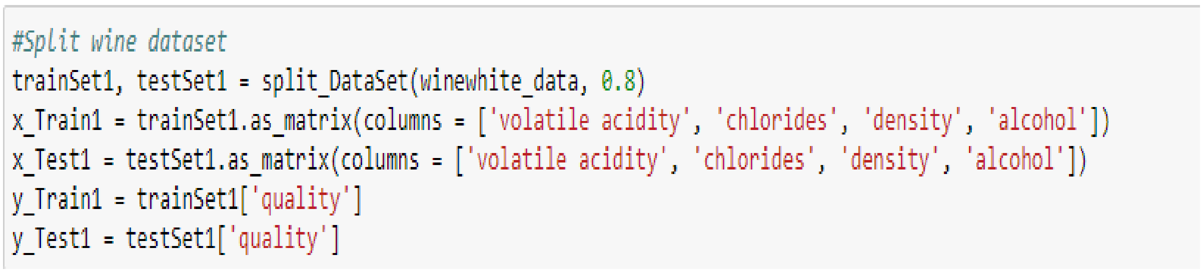
Iris Dataset



```
trainSet, testSet = split_DataSet(iris_data, 0.7)
xTrain = trainSet.as_matrix(columns = ["sepalLength","sepalWidth", "petalLength", "petalWidth"])
xTest = testSet.as_matrix(columns = ["sepalLength","sepalWidth", "petalLength", "petalWidth"])
yTrain = trainSet["target"].as_matrix()
yTest = testSet["target"].as_matrix()
print(xTrain.shape)
print(xTest.shape)
print(yTrain.shape)
print(yTest.shape)
```

```
(103, 4)
(47, 4)
(103,)
(47,)
```

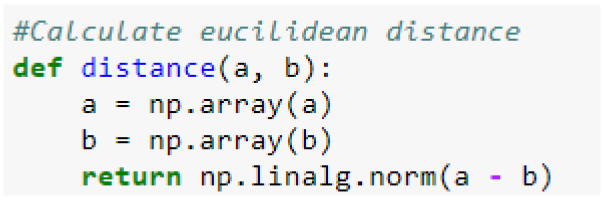
Wine Dataset



```
#Split wine dataset
trainSet1, testSet1 = split_DataSet(winewhite_data, 0.8)
x_Train1 = trainSet1.as_matrix(columns = ['volatile acidity', 'chlorides', 'density', 'alcohol'])
x_Test1 = testSet1.as_matrix(columns = ['volatile acidity', 'chlorides', 'density', 'alcohol'])
y_Train1 = trainSet1['quality']
y_Test1 = testSet1['quality']
```

-Implement a similarity (or a distance) measure. To begin with you can implement the Euclidean Distance

The Eucilidean distance is calculated by



```
#Calculate eucilidean distance
def distance(a, b):
    a = np.array(a)
    b = np.array(b)
    return np.linalg.norm(a - b)
```

-Implement a function that returns top K Nearest Neighbors for a given query and You should provide the prediction for a given query.

```
#Get the nearest neighbours
def get_knn(xTest,target,ins,k,distance=distance):
    xdis = []
    for i in range(len(xTest)):
        dist = distance(ins, xTest[i])
        xdis.append((xTest[i], dist, target[i]))
    xdis.sort(key=lambda x: x[1])
    nn = xdis[:k]
    return nn
#Get the majority voting for the class of nearest neighbours
def vote_knn(nn):
    c = Counter()
    for nbr in nn:
        c[int(nbr[2])] += 1
    return c.most_common(1)[0][0]
```

-Measure the quality of your prediction.

```
#Test the quality of predictions
def accuracy(yTest,yPred):
    count = 0
    for index,i in enumerate(yTest):
        if i != yPred[index]:
            count += 1
    return (len(yTest) - count)/len(yTest)
```

Exercise 2: Optimize and Compare KNN algorithm.

Determine Optimal Value of K in KNN algorithm.

-How you can choose value of K for KNN. Give a criterion to choose an optimal value of K

In KNN, finding the value of k is not easy. A small value of k means that noise will have a higher influence on the result and a large value make it computationally expensive. Data scientists usually choose as an odd number if the number of classes is 2 and another simple approach to select k is set k=sqrt(n).Here I have performed grid search to find the optimal value of K.

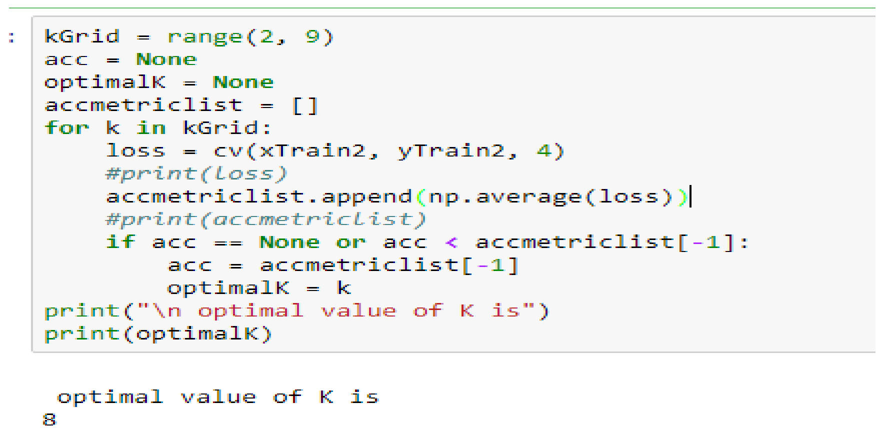
- Implement the criterion for choosing the optimal value of K

Iris Dataset

```
kGrid = range(7, 15)
acc = None
optimalK = None
accmetriclist = []
for k in kGrid:
    loss = cv(xTrain, yTrain, 4)
    #print(loss)
    accmetriclist.append(np.average(loss))
    #print(accmetriclist)
    if acc == None or acc < accmetriclist[-1]:
        acc = accmetriclist[-1]
        optimalK = k
print("\n optimal value of K is")
print(optimalK)
```

```
 optimal value of K is
9
```

Wine Dataset



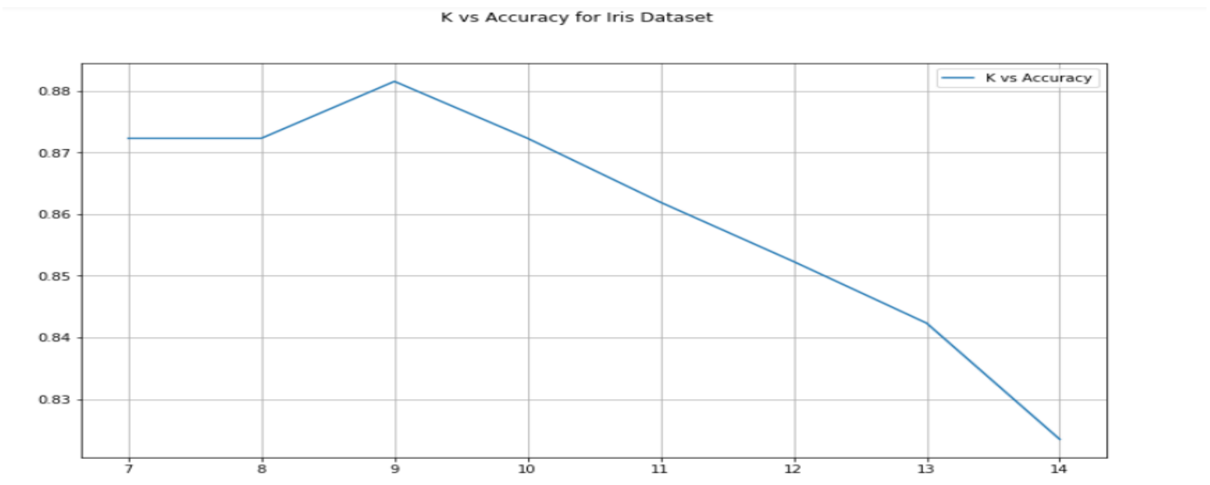
```
: kGrid = range(2, 9)
  acc = None
  optimalK = None
  accmetriclist = []
  for k in kGrid:
      loss = cv(xTrain2, yTrain2, 4)
      #print(loss)
      accmetriclist.append(np.average(loss))
      #print(accmetriclist)
      if acc == None or acc < accmetriclist[-1]:
          acc = accmetriclist[-1]
          optimalK = k
  print("\n optimal value of K is")
  print(optimalK)
```

```
 optimal value of K is
8
```

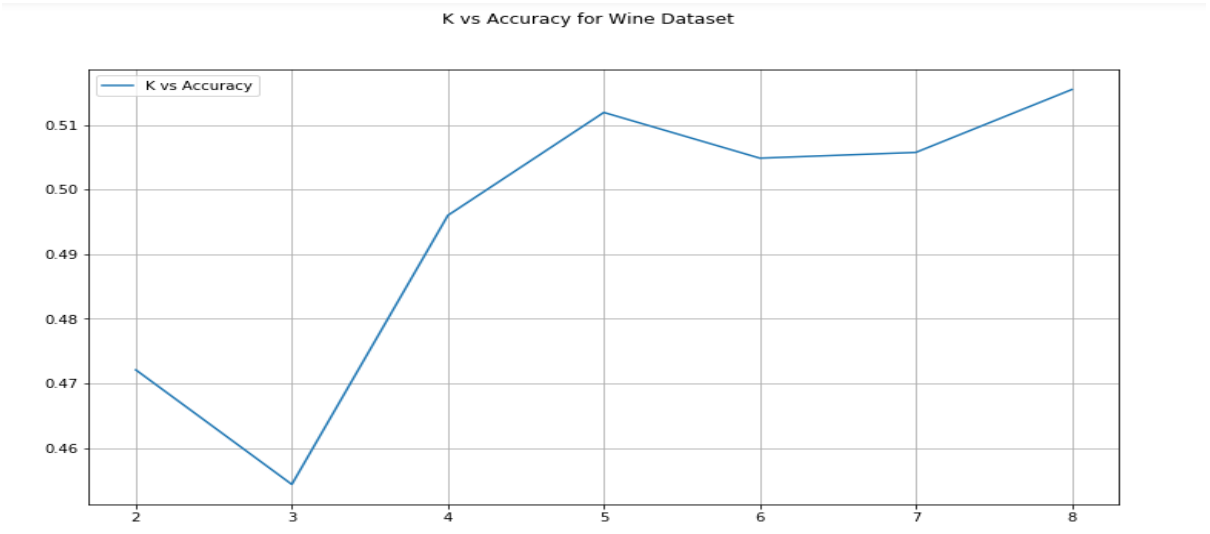
-Experimentally, give evidence that your chosen value is better than other values of K.

Plots of Accuracy with respect to K value

Iris Dataset



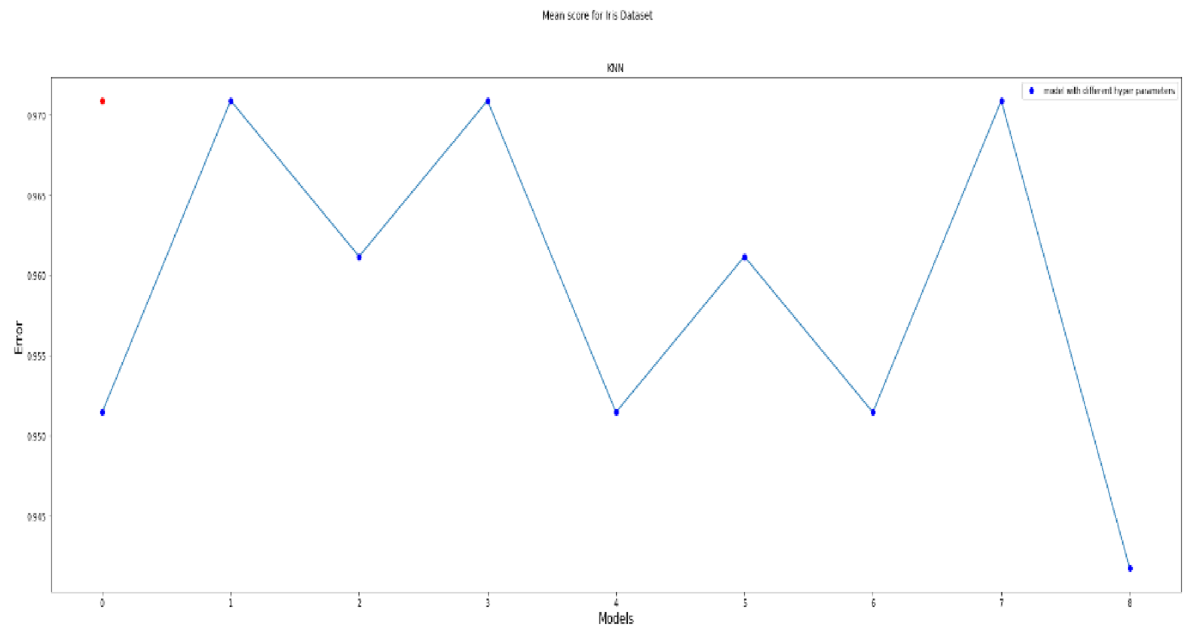
Wine Dataset



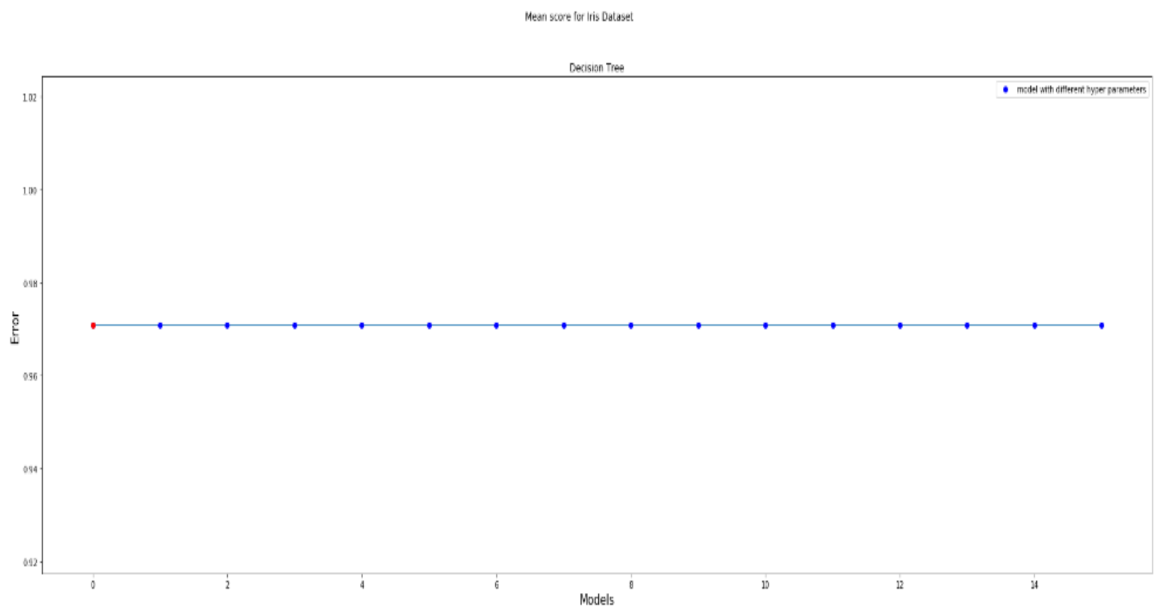
Compare KNN algorithm with Tree based method using Sklearn.

Plots for KNN and Tree based methods

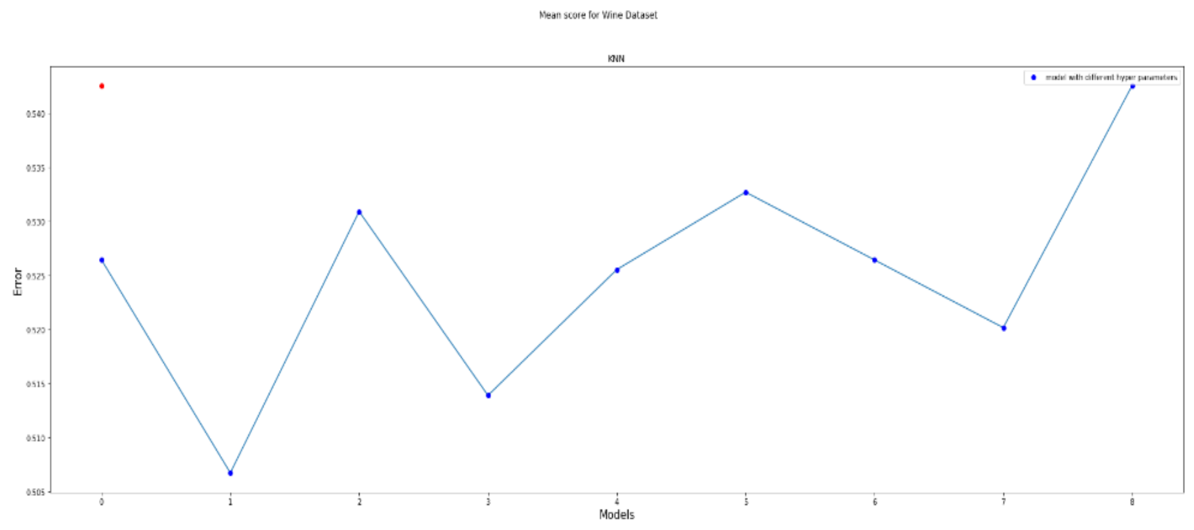
Iris Dataset with KNeighborsClassifier()



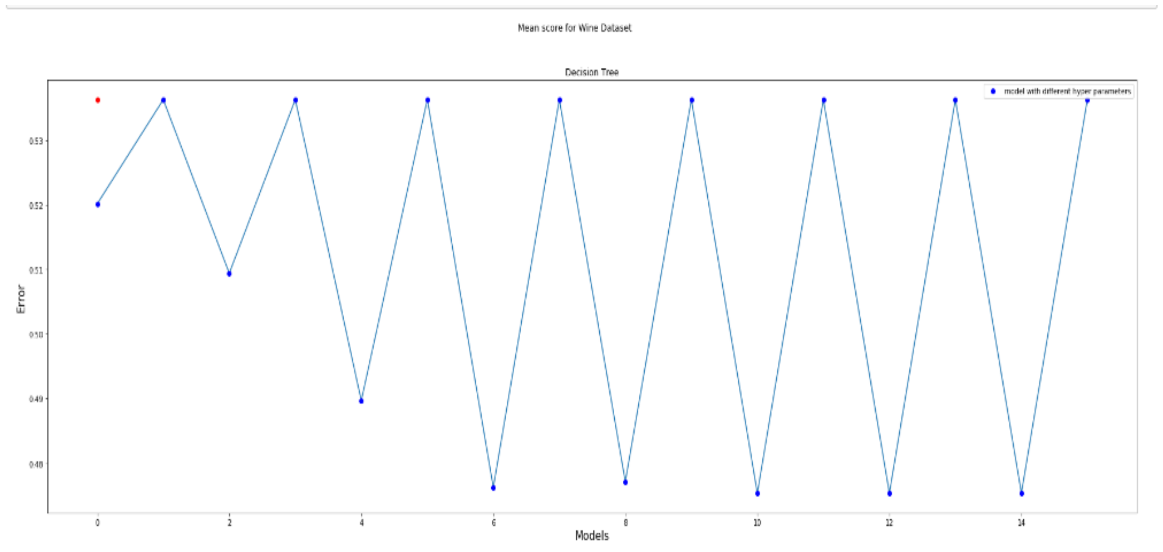
Iris Dataset with DecisionTreeClassifier ()



Wine Dataset with KNeighborsClassifier()



Iris Dataset with DecisionTreeClassifier ()



For the Iris dataset the error is very low in DT compared to KNN and for the wine dataset the error is more or less similar in both algorithms. The main difference somehow is about the domain of application. KNN is used for continuous value inputs, unlike Decision Trees that is applicable for continuous and categorical inputs.