CS57800: Statistical Machine Learning

Homework 0

Vineeth Ravi

September 19, 2018

PU ID : 0030019456

EMAIL ID : ravi24@purdue.edu

1 Introduction

The objective of this assignment is to experiment on Decision Tree classifier and KNN classifier. We will predict the quality of white wine in a scale of 0 to 10 using these two classification algorithms and compare the results. We implement this assignment from scratch in Python 2.7 and we do not use any already implemented models like those in the scikit-learn library.
We include the plots and describe the results and procedure used below :-

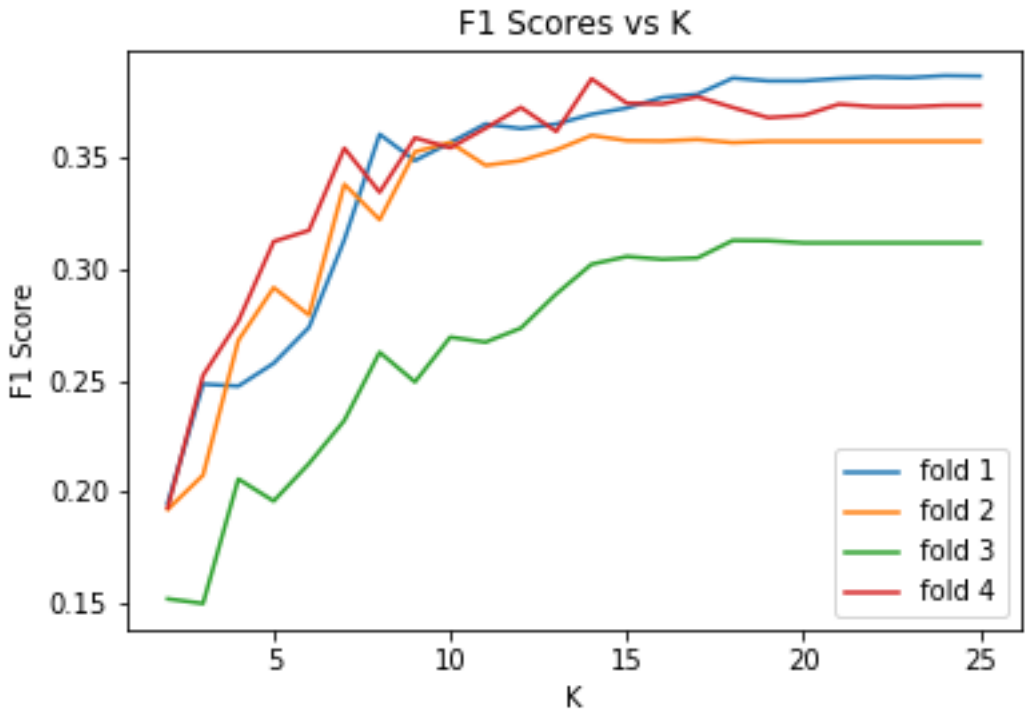
2 Decision Trees

Fold	Max-Depth	Type	F1 Score	Accuracy
1	14	Training	0.970334	0.950340
1	14	Validation	0.390407	0.589918
1	14	Test	0.379104	0.586601
2	8	Training	0.563804	0.660143
2	8	Validation	0.324779	0.538147
2	8	Test	0.291067	0.523077
3	18	Training	0.997058	0.994218
3	18	Validation	0.332892	0.577657
3	18	Test	0.334015	0.562092
4	14	Training	0.913449	0.877468
4	14	Validation	0.406051	0.585831
4	14	Test	0.297040	0.546493

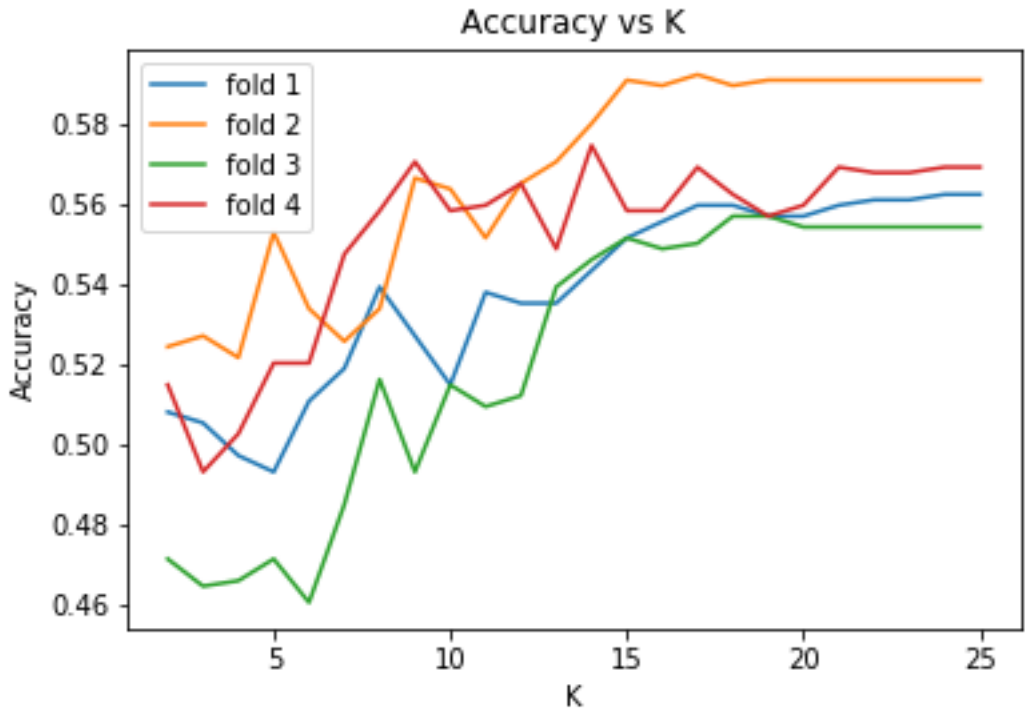
From Various Runs, using different Shuffles , We got various Graphs for F1 score vs Max Depth (K), Analyzing the Graphs, and validation accuracies, The Final-Max Depth is fixed to be 14 in the final output code.

Graphs:-

1) F1 Score vs Max-Depth (Max-Depth = K)



2) Accuracy vs Max-Depth (Max-Depth = K)



3 K-Nearest Neighbours

Fold	K	Type	F1 Score	Accuracy
1	21	Validation	: 0.469644	0.682561
1	21	Test	: 0.384592	0.622549
2	4	Validation	0.356576	0.614441
2	4	Test	0.385745	0.638072
3	25	Validation	0.461426	0.678474
3	25	Test	0.387786	0.654412
4	11	Validation	0.431689	0.652929
4	11	Test	0.385167	0.641279

We observe the graphs of Validation F1 Score vs K, and by running several times on shuffled data, The Hyper-parameters are chosen to be:
Best FINAL K is chosen to be : 16 . We can also choose 11 to be the best.
Distance Metric is chosen to be : Manhattan Distance.

I chose 16, because the best Average Validation F1 : 0.4328 and Test F1: 0.4126. This shuffle data is attached along with the submitted files. This can be loaded if needed to verify the results I have included in the report. Since I ran many shuffled data, to finalize the best metrics and hyper parameters , I have not included all of them in the report.

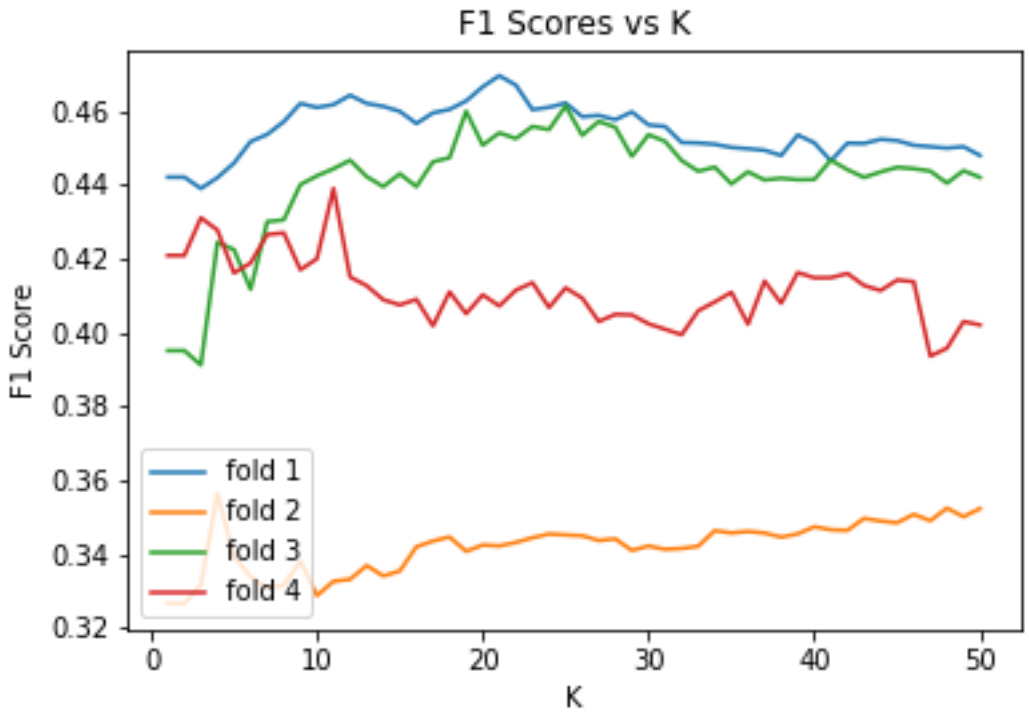
I have shown the best results in the Additional Works section, which has the best F1 and accuracy scores for test and validation, across many of my runs. The data file for that run, is included with the files for verifying the numbers mentioned in the report.

Final Comments

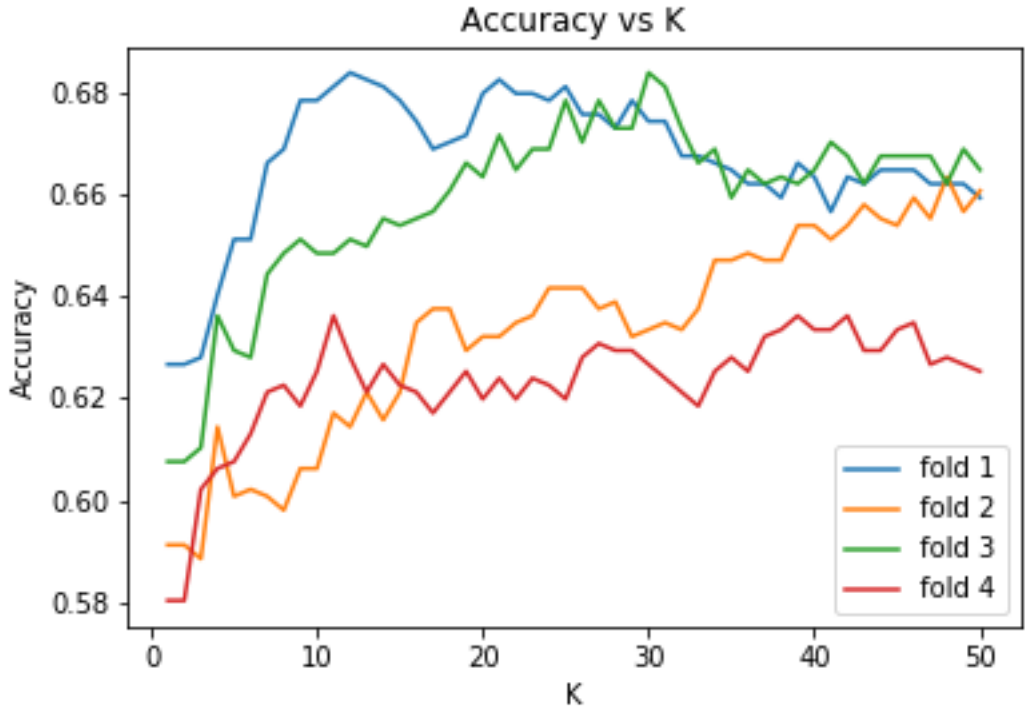
For Both Decision Trees and KNN The Hyper Parameters were chosen purely from the Validation F1 Score and Accuracy Graph results. We tune them, to maximize Validation F1 Score. Hence from the graphs we choose the best values of K, Distance Metrics, Max-Depth. We do not just run one time for 1 shuffled data set. I have run and obtained 5 sets of graphs - for 5 different shuffles of the data (all of them are not included in this report, (only 1 of the runs based on the random.seed(0) is include) - This can be changed in the code as well. I can send all the graphs for every shuffled data, I generated if they have extra bonus points. Based on comparison of the results (Validation F1 Score) across these 5 runs, The final results are:- K value is chosen to be 16, Max-Depth = 14 , Distance = Manhattan (All of Not, necessarily obtained from the graphs above), But from the other graph runs on shuffled data sets, As shown in the additional works section.

Graphs:-
a) Manhattan Distance

1) F1 Score vs K

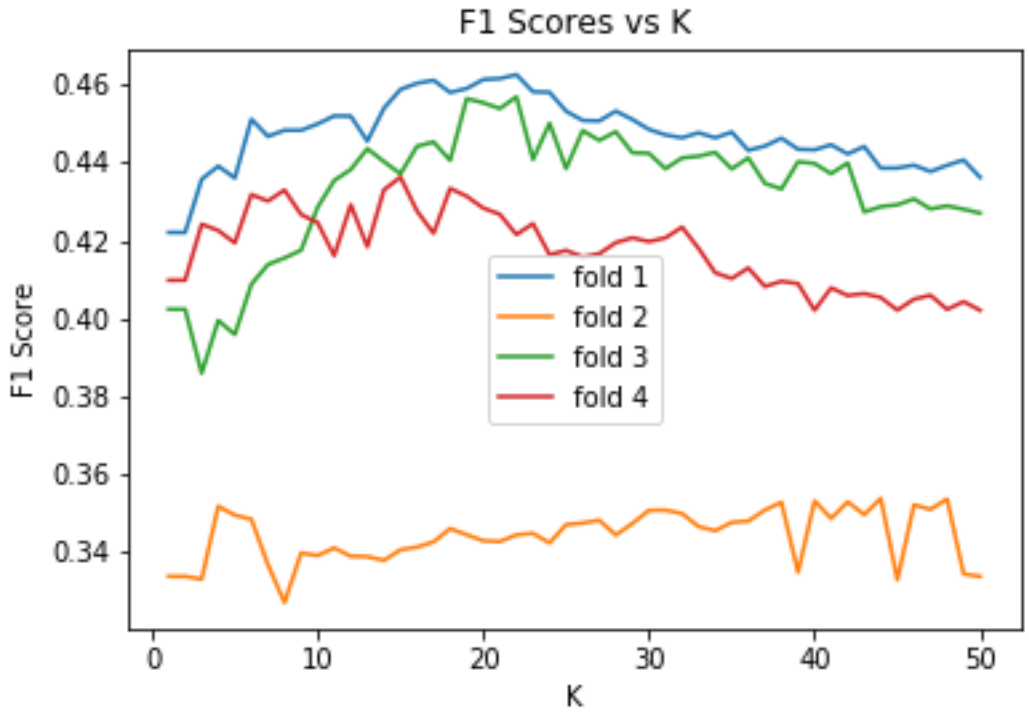


2) Accuracy vs K

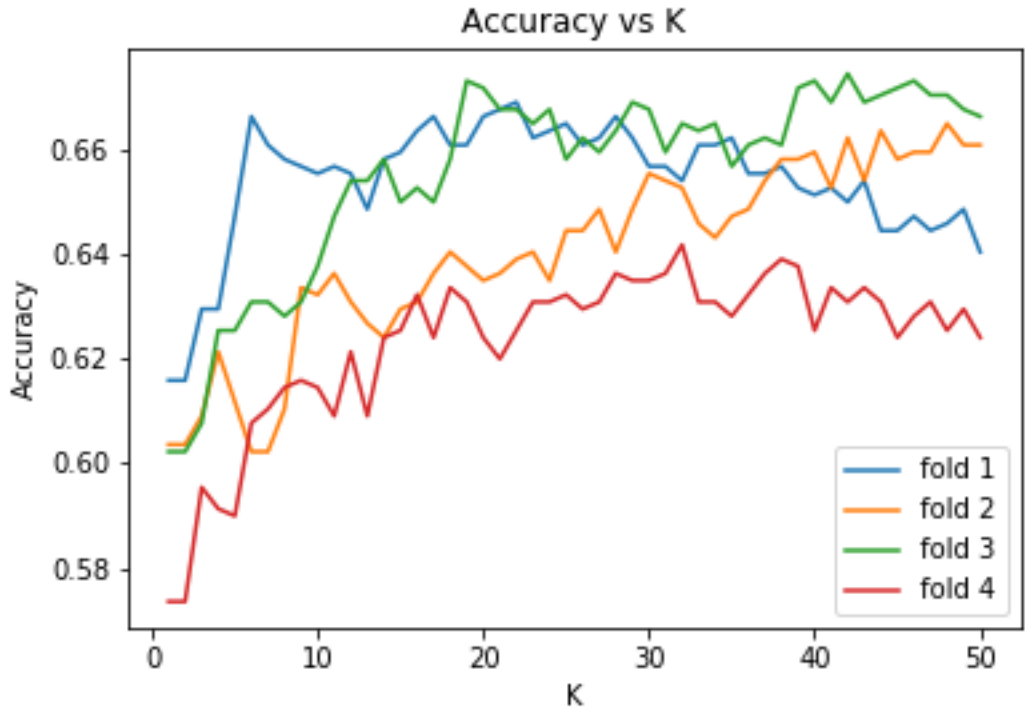


b) Euclidian Distance

1) F1 Score vs K

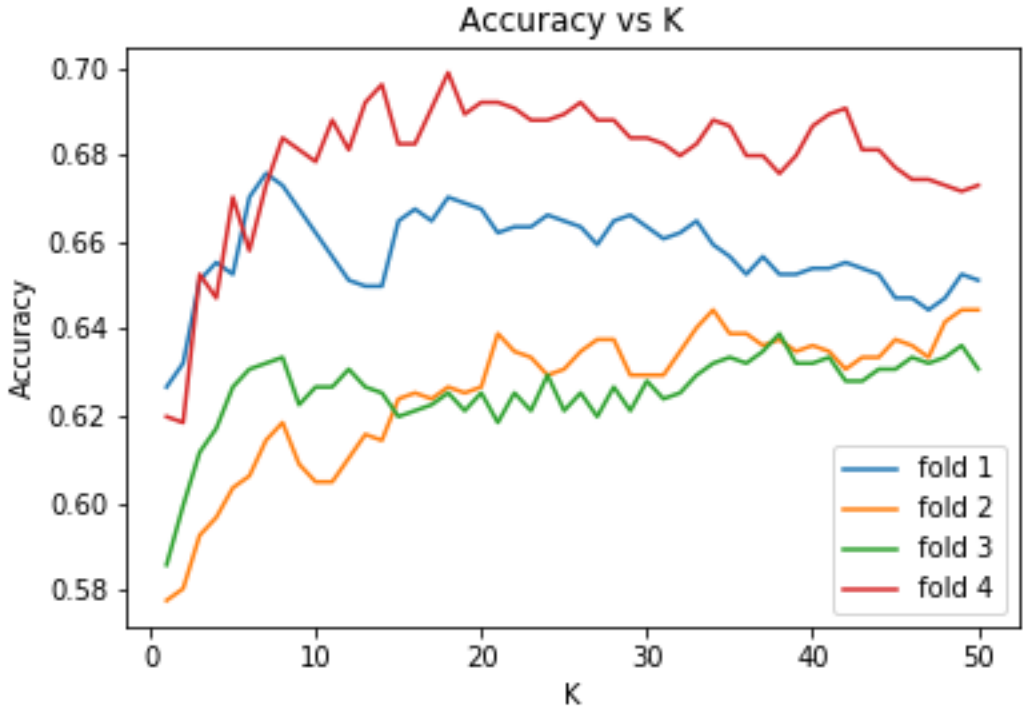


2) Accuracy vs K

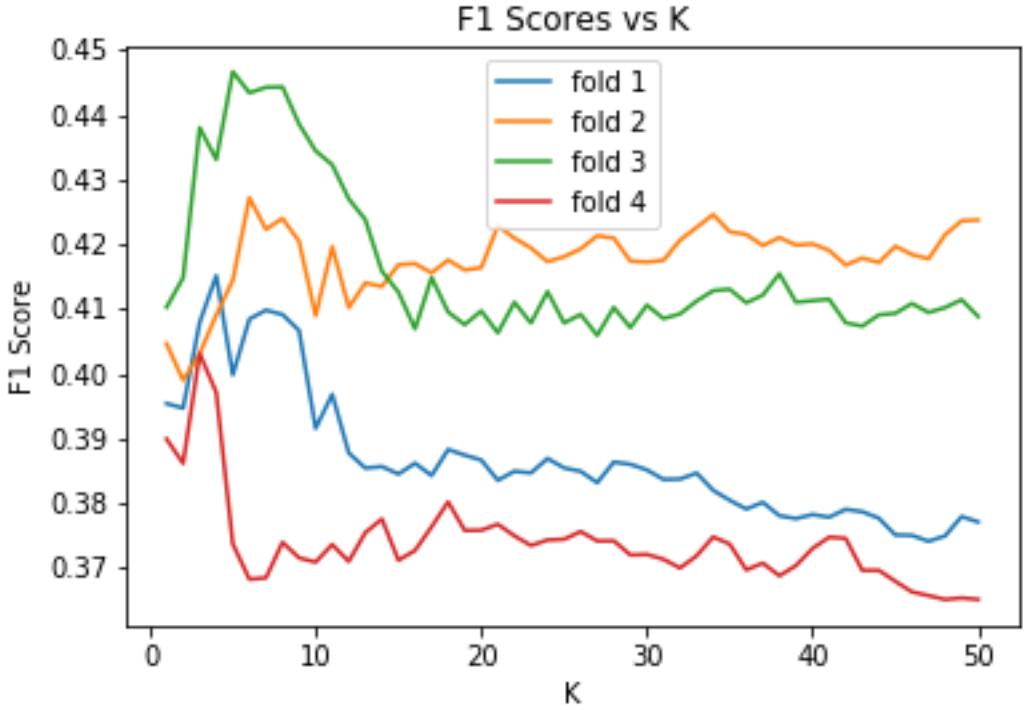


c) Cosine Distance

1) F1 Score vs K



2) Accuracy vs K



4 Answers

1) Answer :- Over-fitting and Lack of Generalization. When we allow the Max-Depth upto the number of features in a decision tree, Then we would be over-fitting the data to the training set and we would not be able to generalize easily to new test data points. We would then be creating a very large tree, and would need to spend more time on post-pruning techniques to avoid overfitting the data.
2)

a) Decision Trees, the training step is the most expensive and time consuming step, Since most computation on the training data happens for generating the decision tree, whereas for KNN's it is a type of lazy learning process, where training literally has no computation cost, apart from just storing the training data in memory.

b) The decision trees divides the feature space into axis parallel rectangles, where as using KNN, we get non-linear decision boundaries. The KNN cannot take categorical data as input features, since we compute distance metrics, But using decision trees, we can include categorical features as well, using Key: Values, as mentioned in the Additional Works Section.

c) The computation time, to evaluate a test data point for KNN is expensive, but for a decision tree, it is very easy and fast to predict the label of a test data point.

3) To convert your decision tree from a classification model to a ranking model, We can :-
a) Use the weighted average of the class labels present at a given leaf node, when we make a prediction instead of taking the argmax(of class labels) present in the leaf node.
b) When we use post pruning or max-depth to grow the tree, we can use a top K-gini impurity measure to keep track of the ranking error, when we make the prediction of labels at a given node. We can use this measure instead of Majority Error, to classify and rank the prediction of labels possible at a given node. This will help in converting the decision tree from a classification to a ranking model.

5 Scope of Creativity

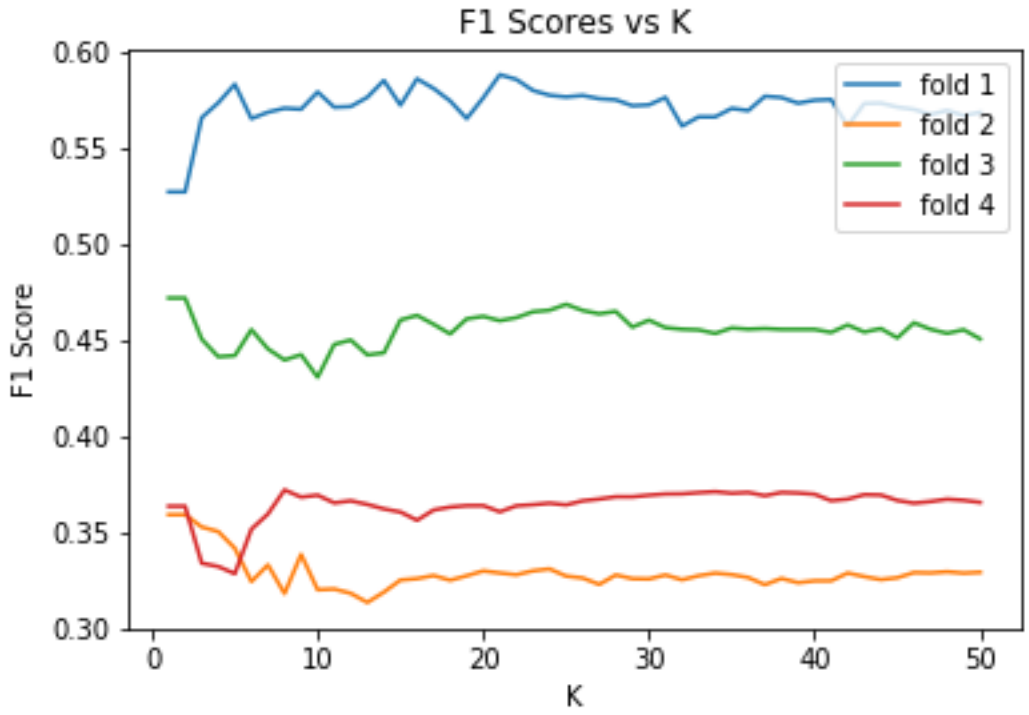
We can replace missing data values, with either the median or the mode of the data columns for that particular feature. We can normalize them using a Gaussian or sigmoid , Set thresholds by dividing them into a prefixed number of categories, which are used, when we create the tree. This is how , I would deal with the problem of missing data - using mode and median values in the corresponding feature column, to replace and fill them. Gaussian Seems to work better than sigmoid, for getting higher F1-Scores, because of better normalization.

6 Additional Works : Bonus Points

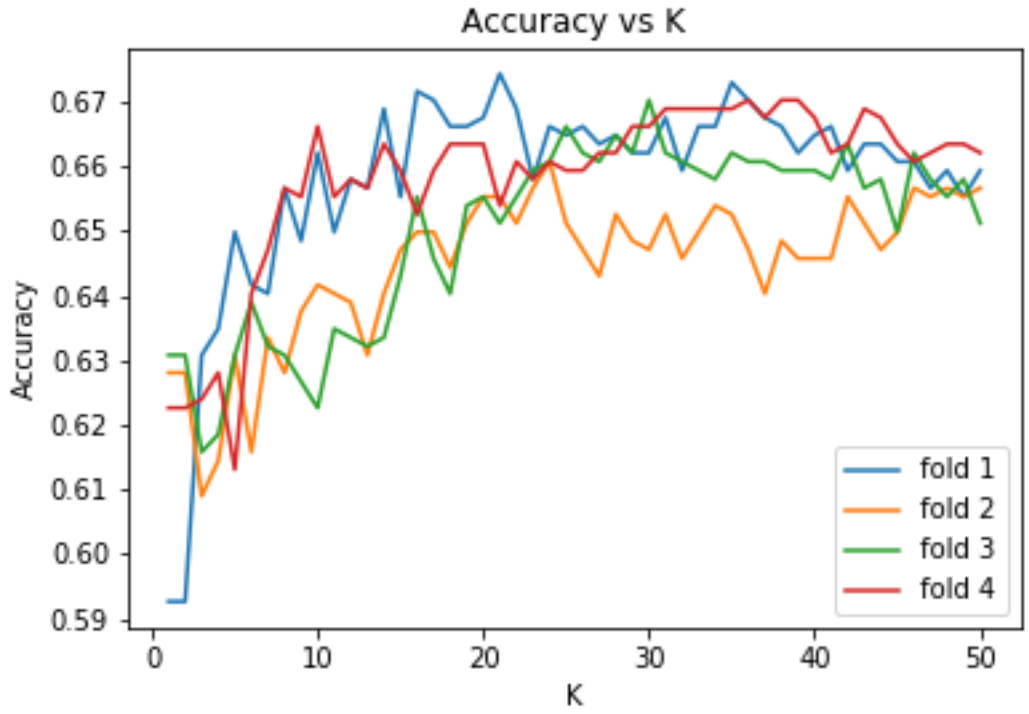
Best Results using KNN :-
K = 16 , Manahattan Distance
Average Test F1 Score is : 0.4126
Average Validation F1 score is : 0.4328

Best Results

1) F1 Score vs K



2) Accuracy vs K



The Euclidean distance works best with K = 19. This can be modified in my code, by changing the call to the distance function.

K = 19 ,Euclidean Distance
Average Test F1 Score is : 0.3863
Average Validation F1 score is : 0.3988

POST PRUNING in Decision trees:-
The variable in the code min instances present in node, is increased from 2 to 10, This helps reduced the max-depth of the tree. This is implemented as a post pruning process, and can be tuned again as a hyper parameter in the code. For example from 2 to 10.

Comparison of several distance metrics and tuning of hyperparameters in KNN.
Making several features as categorical from continuous variables were tried, But they do not yield the best results, which are finally included in the code.

7 Python Code

7.1 Decision Trees - Cross Validation Pointer

```
def cross_validation_test(X_train,K_Max):
    X_val=X_train[0:np.shape(X_train)[0]/5]
    X_train=X_train[np.shape(X_train)[0]/5:np.shape(X_train)[0]]
```

```
    Y_train=X_train[:,11]
    Y_val=X_val[:,11]

    Mean=X_train[:,0:11].mean(0)
    Std=X_train[:,0:11].std(0)
    X_train[:,0:11]=(X_train[:,0:11]-Mean)/Std
    X_val[:,0:11]=(X_val[:,0:11]-Mean)/Std

    Accuracy=np.zeros(K_Max+1)
    F1_Score=np.zeros(K_Max+1)

    for k in range(2,K_Max+1):
        Y_pred=np.zeros(len(X_val))
        tree = create_decision_treeID3(X_train,min_instance_count=2,Max_Depth=k,de
        for i in range(0,len(X_val)):
            Y_pred[i]=make_decision(X_val[i],tree)

        Accuracy[k],F1_Score[k]=CM(Y_pred,Y_val)

        print("The value of Max-Depth is %d ." %(k))
        print(F1_Score[k])

    return Accuracy,F1_Score
```

7.2 K-Nearest Neighbours - Cross Validation Pointer

```
def cross_validation_test(X_train,K_Max):
    X_val=X_train[0:np.shape(X_train)[0]/5]
    X_train=X_train[np.shape(X_train)[0]/5:np.shape(X_train)[0]]

    Y_train=X_train[:,11]
    X_train=X_train[:,0:-1]
    Y_val=X_val[:,11]
    X_val=X_val[:,0:-1]

    Mean=X_train.mean(0)
    Std=X_train.std(0)
    X_train=(X_train-Mean)/Std
    X_val=(X_val-Mean)/Std

    Accuracy=np.zeros(K_Max+1)
    F1_Score=np.zeros(K_Max+1)
```

```
    for k in range(1,K_Max+1):
        Y_pred=np.zeros(len(X_val))
        for i in range(0,len(X_val)):
            distance=[]
            Wts=np.zeros(11)

            for tmp in range(0,k):
                d=manhattan(X_train[tmp],X_val[i])
                l=[Y_train[tmp],d]
                distance.append(l)

            for j in range(k,len(X_train)):
                d=manhattan(X_train[j],X_val[i])
                tmp=np.argmax(np.asarray(distance)[:,1])
                if(d<distance[tmp][1]):
                    del distance[tmp]
                    l=[Y_train[j],d]
                    distance.append(l)

            for m in range(0,k):
                Wts[int(distance[m][0])]=Wts[int(distance[m][0])]+np.abs(1/float(di

            tmp=np.argmax(Wts)
            Y_pred[i]=tmp

        Accuracy[k],F1_Score[k]=CM(Y_pred,Y_val)

        print("The value of K is %d ." %(k))
        print("The F1 Score is %f ." %(F1_Score[k]))

    return Accuracy,F1_Score
```

7.3 K-Nearest Neighbours

```
import csv
import sys
import time
import numpy as np
#import matplotlib.pyplot as plt

#Remove Above Comment Line for getting the graphs included in reports like F1 vs K

start = time.time()
```

```
def euclidian(a,b):
    dist=np.linalg.norm(a-b)
    if(dist==0):
        dist=0.00001

    return np.abs(dist)

def cosine(a,b):
    dist=1-(np.dot(a,b)/(np.linalg.norm(a)*np.linalg.norm(b)))
    if(dist==0):
        dist=0.00001

    return np.abs(dist)

def manhattan(a,b):
    dist=np.sum(np.abs(a-b))
    if(dist==0):
        dist=0.00001

    return np.abs(dist)

def CM(Y_pred,Y_true):
    Con_Mat=np.zeros((11,11))
    TP=np.zeros(11)
    FP=np.zeros(11)
    FN=np.zeros(11)
    F=np.zeros(11)

    for i in range(0,len(Y_pred)):
        Con_Mat[int(Y_true[i])][int(Y_pred[i])]=Con_Mat[int(Y_true[i])][int(Y_pred

    for i in range(0,11):
        for j in range(0,11):
            if(i==j):
                TP[i]=Con_Mat[i][j]
            else:
                FN[i]=FN[i]+Con_Mat[i][j]
                FP[i]=FP[i]+Con_Mat[j][i]
        if(TP[i]==0):
            F[i]=0
        else:
            F[i]=2*TP[i]/float(2*TP[i]+FP[i]+FN[i])
```

```
    F1_Score=float(np.sum(F))/(len(np.unique(Y_true)))
    Accuracy=float(np.sum(TP))/(len(Y_pred))

    return Accuracy,F1_Score

# THE CROSS VALIDATION TASK - SPECIAL POINTER TO THE BEGINING OF THE CODE FUNCTION
def cross_validation_test(X_train,K_Max):
    X_val=X_train[0:np.shape(X_train)[0]/5]
    X_train=X_train[np.shape(X_train)[0]/5:np.shape(X_train)[0]]

    Y_train=X_train[:,11]
    X_train=X_train[:,0:-1]
    Y_val=X_val[:,11]
    X_val=X_val[:,0:-1]

    Mean=X_train.mean(0)
    Std=X_train.std(0)
    X_train=(X_train-Mean)/Std
    X_val=(X_val-Mean)/Std

    Accuracy=np.zeros(K_Max+1)
    F1_Score=np.zeros(K_Max+1)

    for k in range(1,K_Max+1):
        Y_pred=np.zeros(len(X_val))
        for i in range(0,len(X_val)):
            distance=[]
            Wts=np.zeros(11)

            for tmp in range(0,k):
                d=manhattan(X_train[tmp],X_val[i])
                l=[Y_train[tmp],d]
                distance.append(l)

            for j in range(k,len(X_train)):
                d=manhattan(X_train[j],X_val[i])
                tmp=np.argmax(np.asarray(distance)[:,1])
                if(d<distance[tmp][1]):
                    del distance[tmp]
                    l=[Y_train[j],d]
                    distance.append(l)

            for m in range(0,k):
```

```
                Wts[int(distance[m][0])]=Wts[int(distance[m][0])]+np.abs(1/float(di

            tmp=np.argmax(Wts)
            Y_pred[i]=tmp

        Accuracy[k],F1_Score[k]=CM(Y_pred,Y_val)

        print("The value of K is %d ." %(k))
        print("The F1 Score is %f ." %(F1_Score[k]))

    return Accuracy,F1_Score


def Final_Output_Test(X_train,X_test,K):
    X_val=X_train[0:np.shape(X_train)[0]/5]
    X_train=X_train[np.shape(X_train)[0]/5:np.shape(X_train)[0]]

    Y_train=X_train[:,11]
    X_train=X_train[:,0:-1]
    Y_val=X_val[:,11]
    X_val=X_val[:,0:-1]
    Y_test=X_test[:,11]
    X_test=X_test[:,0:-1]

    Mean=X_train.mean(0)
    Std=X_train.std(0)
    X_train=(X_train-Mean)/Std
    X_val=(X_val-Mean)/Std
    X_test=(X_test-Mean)/Std

    for k in range(K,K+1):
        Y_pred=np.zeros(len(X_val))
        for i in range(0,len(X_val)):
            distance=[]
            Wts=np.zeros(11)

            for tmp in range(0,k):
                d=manhattan(X_train[tmp],X_val[i])
                l=[Y_train[tmp],d]
                distance.append(l)

            for j in range(k,len(X_train)):
                d=manhattan(X_train[j],X_val[i])
```

```
            tmp=np.argmax(np.asarray(distance)[:,1])
            if(d<distance[tmp][1]):
                del distance[tmp]
                l=[Y_train[j],d]
                distance.append(l)

        for m in range(0,k):
            Wts[int(distance[m][0])]=Wts[int(distance[m][0])]+np.abs(1/float(di

        tmp=np.argmax(Wts)
        Y_pred[i]=tmp

    VAccuracy,VF1_Score=CM(Y_pred,Y_val)


for k in range(K,K+1):
    Y_pred=np.zeros(len(X_test))
    for i in range(0,len(X_test)):
        distance=[]
        Wts=np.zeros(11)

        for tmp in range(0,k):
            d=manhattan(X_train[tmp],X_test[i])
            l=[Y_train[tmp],d]
            distance.append(l)

        for j in range(k,len(X_train)):
            d=manhattan(X_train[j],X_test[i])
            tmp=np.argmax(np.asarray(distance)[:,1])
            if(d<distance[tmp][1]):
                del distance[tmp]
                l=[Y_train[j],d]
                distance.append(l)

        for m in range(0,k):
            Wts[int(distance[m][0])]=Wts[int(distance[m][0])]+np.abs(1/float(di

        tmp=np.argmax(Wts)
        Y_pred[i]=tmp

    TAccuracy,TF1_Score=CM(Y_pred,Y_test)

return TAccuracy,TF1_Score,VAccuracy,VF1_Score
```

```
file = open('winequality-white.csv')
data=[]

TAc=[]
TF1=[]
VAc=[]
VF1=[]

for row in file:
    a=row.split(';')
    data.append(a)

del data[0]

X=np.asarray(data).astype('float')
np.random.seed(0)
np.random.shuffle(X)

X_1=X[0:np.shape(X)[0]/4]
X_2=X[np.shape(X)[0]/4:2*(np.shape(X)[0]/4)]
X_3=X[2*(np.shape(X)[0]/4):3*(np.shape(X)[0]/4)]
X_4=X[3*(np.shape(X)[0]/4):np.shape(X)[0]]

test=[X_1,X_2,X_3,X_4]
tr1=np.concatenate((X_2,X_3,X_4),axis=0)
tr2=np.concatenate((X_3,X_4,X_1),axis=0)
tr3=np.concatenate((X_4,X_1,X_2),axis=0)
tr4=np.concatenate((X_1,X_2,X_3),axis=0)
train=[tr1,tr2,tr3,tr4]

K_best_final=21
#4,11
print("Hyper-parameters:")
print("K : %d" %(K_best_final))
print("Distance Measure: Manhattan Distance")

for i in range(0,4):

    X_test=test[i]
    X_train=train[i]
```

```
    TAccuracy,TF1_Score,VAccuracy,VF1_Score=Final_Output_Test(X_train,X_test,K_bes
    TAc.append(TAccuracy)
    TF1.append(TF1_Score)
    VAc.append(VAccuracy)
    VF1.append(VF1_Score)

    print("Hyper-parameters:")
    print("K Best Fold: %d" %(K_best_fold))
    print("Distance Measure: Manhattan Distance")

    print("Fold-%d:" %(i+1))
    print("Validation: F1 Score: %f , Accuracy: %f" %(VF1_Score,VAccuracy))
    print("Test: F1 Score: %f , Accuracy: %f" %(TF1_Score,TAccuracy))

print("Average:")
print("Validation: F1 Score: %f , Accuracy: %f" %(np.mean(VF1),np.mean(VAc)))
print("Test: F1 Score: %f , Accuracy: %f" %(np.mean(TF1),np.mean(TAc)))

end = time.time()
print("The time taken for the algorithm computation is :- %f seconds." % (end-start

# The END OF CODE which Prints the FINAL REQUIRED OUTPUT with the Best K Final Valu

# The Below Code is for Running the Cross Validation & Testing
# That is for running the cross_validate_test function
# Comment the Above Main Section Code
# Un Comment The Below Section, to just run 4-Fold Cross Validation
# The below section is used to generate the graphs included in the report
# By Running the Below Code we tune the hyperparamet K_best_final
# We also tune the distance metric, by changing the function we call :
# Euclidian, Cosine, Manhattan are the distance hyperparameters to tune.

#file = open('winequality-white.csv')
#data=[]
#Ac=[]
#F1=[]
#
#TAc=[]
#TF1=[]
#VAc=[]
#VF1=[]
#
#for row in file:
```

```
#     a=row.split(';')
#     data.append(a)
#
#del data[0]
#
#X=np.asarray(data).astype('float')
#np.random.seed(0)
#np.random.shuffle(X)
#
#X_1=X[0:np.shape(X)[0]/4]
#X_2=X[np.shape(X)[0]/4:2*(np.shape(X)[0]/4)]
#X_3=X[2*(np.shape(X)[0]/4):3*(np.shape(X)[0]/4)]
#X_4=X[3*(np.shape(X)[0]/4):np.shape(X)[0]]
#
#test=[X_1,X_2,X_3,X_4]
#tr1=np.concatenate((X_2,X_3,X_4),axis=0)
#tr2=np.concatenate((X_3,X_4,X_1),axis=0)
#tr3=np.concatenate((X_4,X_1,X_2),axis=0)
#tr4=np.concatenate((X_1,X_2,X_3),axis=0)
#train=[tr1,tr2,tr3,tr4]
#
#for i in range(0,4):
#
#     X_test=test[i]
#     X_train=train[i]
#
#     Accuracy,F1_Score=cross_validation_test(X_train,50)
#     K_best_fold=np.argmax(F1_Score)
#     print("The best value of K is %d and fold number is %d." % (K_best_fold,i+1))
#     print(Accuracy[K_best_fold])
#     print(F1_Score[K_best_fold])
#
#     x = np.arange(1,51, 1)
#     Accuracy=Accuracy[1:]
#     F1_Score=F1_Score[1:]
#     F1.append(F1_Score)
#     Ac.append(Accuracy)
#
#     plt.figure(1)
#     plt.plot(x,Accuracy, label = "fold %d" %(i+1))
#     plt.figure(2)
#     plt.plot(x,F1_Score, label = "fold %d" %(i+1))
#
```

```
#     TAccuracy,TF1_Score,VAccuracy,VF1_Score=Final_Output_Test(X_train,X_test,K_be
#     TAc.append(TAccuracy)
#     TF1.append(TF1_Score)
#     VAc.append(VAccuracy)
#     VF1.append(VF1_Score)
#
#     print("Hyper-parameters:")
#     print("K Best Fold: %d" %(K_best_fold))
#     print("Distance Measure: Manhattan Distance")
#
#     print("Fold-%d:" %(i+1))
#     print("Validation: F1 Score: %f , Accuracy: %f" %(VF1_Score,VAccuracy))
#     print("Test: F1 Score: %f , Accuracy: %f" %(TF1_Score,TAccuracy))
#
#
#print("Average:")
#print("Validation: F1 Score: %f , Accuracy: %f" %(np.mean(VF1),np.mean(VAc)))
#print("Test: F1 Score: %f , Accuracy: %f" %(np.mean(TF1),np.mean(TAc)))
#
#
#plt.figure(1)
#plt.xlabel('K')
## naming the y axis
#plt.ylabel('Accuracy')
## giving a title to my graph
#plt.title('Accuracy vs K')
## show a legend on the plot
#plt.legend()
## function to show the plot
#plt.savefig('Accuracy.png')
#
#plt.figure(2)
#plt.xlabel('K')
## naming the y axis
#plt.ylabel('F1 Score')
## giving a title to my graph
#plt.title('F1 Scores vs K')
## show a legend on the plot
#plt.legend()
## function to show the plot
#plt.savefig('F1_Score.png')
#
#pickle.dump(X, open( "X_data_saved.p", "wb" ) )
```

```
#pickle.dump(Ac, open( "Ac_data_saved.p", "wb" ) )
#pickle.dump(F1, open( "F1_data_saved.p", "wb" ) )
#
#end = time.time()
#print("The time taken for the algorithm computation is :- %f seconds." % (end-sta
```

7.4 Decision Trees

```
import csv
import sys
import time
import numpy as np
#import matplotlib.pyplot as plt

import pickle


start = time.time()


def unique_data_labels(data):
    count=len(np.unique(data[:,-1]))
    if(count==1):
        return 1
    else:
        return 0

def classify_data_labels(data):
    array,counts=np.unique(data[:,-1],return_counts=True)
    tmp=np.argmax(counts)
    predicted_label=array[tmp]

    return predicted_label

def data_entropy(data):
    array,counts=np.unique(data[:,-1],return_counts=True)
    Pr_array=counts/float(np.sum(counts))
    entropy=-np.sum(Pr_array*(np.log2(Pr_array)))

    return entropy

def expected_entropy(data_left,data_right):
    p_left=float(len(data_left))/(len(data_left)+len(data_right))
    p_right=float(len(data_right))/(len(data_left)+len(data_right))
```

```
    exp_entropy=(p_left*data_entropy(data_left) + p_right*data_entropy(data_right))

    return exp_entropy

def information_gain(data,data_left,data_right):
    e1=data_entropy(data)
    e2=expected_entropy(data_left,data_right)
    IG=e1-e2

    return IG

def create_data_split(data,split_col,split_val):
    split_col_values=data[:,split_col]
    data_left=data[split_col_values<=split_val]
    data_right=data[split_col_values>split_val]

    return data_left,data_right

def find_best_decision(data):
    split_choices={}

    for i in range(0,np.shape(data)[1]-1):
        split_choices[i]=[]
        unique_values=np.unique(data[:,i])
        for j in range(0,len(unique_values)):
            if(j!=0):
                val1=unique_values[j]
                val2=unique_values[j-1]
                split_val=(val1+val2)/2
                split_choices[i].append(split_val)

    IG=0

    for i in range(0,len(split_choices)):
        for j in split_choices[i]:
            data_left,data_right=create_data_split(data,i,j)
            IG_new=information_gain(data,data_left,data_right)
            if(IG_new>=IG):
                IG=IG_new
                best_split_col=i
                best_split_val=j

    return best_split_col,best_split_val
```

```
def CM(Y_pred,Y_true):
    Con_Mat=np.zeros((11,11))
    TP=np.zeros(11)
    FP=np.zeros(11)
    FN=np.zeros(11)
    F=np.zeros(11)

    for i in range(0,len(Y_pred)):
        Con_Mat[int(Y_true[i])][int(Y_pred[i])]=Con_Mat[int(Y_true[i])][int(Y_pred[

    for i in range(0,11):
        for j in range(0,11):
            if(i==j):
                TP[i]=Con_Mat[i][j]
            else:
                FN[i]=FN[i]+Con_Mat[i][j]
                FP[i]=FP[i]+Con_Mat[j][i]
        if(TP[i]==0):
            F[i]=0
        else:
            F[i]=2*TP[i]/float(2*TP[i]+FP[i]+FN[i])

    F1_Score=float(np.sum(F))/(len(np.unique(Y_true)))
    Accuracy=float(np.sum(TP))/(len(Y_pred))

    return Accuracy,F1_Score

def create_decision_treeID3(data,min_instance_count=2,Max_Depth=11,depth_count=0):

    if((unique_data_labels(data))or(len(data)<min_instance_count)or(depth_count==M
        predicted_label=classify_data_labels(data)
        return predicted_label

    else:
        depth_count=depth_count+1
        bsplit_col,bsplit_val=find_best_decision(data)
        data_left,data_right=create_data_split(data,bsplit_col,bsplit_val)
        decision="{} <= {}".format(bsplit_col,bsplit_val)
        sub_tree={decision:[]}

        left=create_decision_treeID3(data_left,min_instance_count,Max_Depth,depth_c
        right=create_decision_treeID3(data_right,min_instance_count,Max_Depth,deptl
```

```
        if(left==right):
            sub_tree=left
        else:
            sub_tree[decision].append(left)
            sub_tree[decision].append(right)

        return sub_tree

def make_decision(x,tree):
    decision = list(tree.keys())[0]
    col_index, comparison_operator, value = decision.split(" ")

    if(x[int(col_index)]<=float(value)):
        label=tree[decision][0]
    else:
        label=tree[decision][1]

    if not isinstance(label, dict):
        return label
    else:
        return make_decision(x,label)

def Final_Output_Test(X_train,X_test,K):
    X_val=X_train[0:np.shape(X_train)[0]/5]
    X_train=X_train[np.shape(X_train)[0]/5:np.shape(X_train)[0]]

    Y_train=X_train[:,11]
    Y_val=X_val[:,11]
    Y_test=X_test[:,11]

    Mean=X_train[:,0:11].mean(0)
    Std=X_train[:,0:11].std(0)
    X_train[:,0:11]=(X_train[:,0:11]-Mean)/Std
    X_val[:,0:11]=(X_val[:,0:11]-Mean)/Std
    X_test[:,0:11]=(X_test[:,0:11]-Mean)/Std

    for k in range(K,K+1):
        tree = create_decision_treeID3(X_train,min_instance_count=2,Max_Depth=k,de

        Y_pred=np.zeros(len(X_train))
        for i in range(0,len(X_train)):
            Y_pred[i]=make_decision(X_train[i],tree)
```

```
        TrAccuracy, TrF1_Score=CM(Y_pred, Y_train)

        Y_pred=np.zeros(len(X_val))
        for i in range(0,len(X_val)):
            Y_pred[i]=make_decision(X_val[i],tree)

        VAccuracy, VF1_Score=CM(Y_pred, Y_val)

        Y_pred=np.zeros(len(X_test))
        for i in range(0,len(X_test)):
            Y_pred[i]=make_decision(X_test[i],tree)

        TsAccuracy, TsF1_Score=CM(Y_pred, Y_test)

    return TsAccuracy, TsF1_Score, VAccuracy, VF1_Score, TrAccuracy, TrF1_Score


def cross_validation_test(X_train,K_Max):
    X_val=X_train[0:np.shape(X_train)[0]/5]
    X_train=X_train[np.shape(X_train)[0]/5:np.shape(X_train)[0]]

    Y_train=X_train[:,11]
    Y_val=X_val[:,11]

    Mean=X_train[:,0:11].mean(0)
    Std=X_train[:,0:11].std(0)
    X_train[:,0:11]=(X_train[:,0:11]-Mean)/Std
    X_val[:,0:11]=(X_val[:,0:11]-Mean)/Std

    Accuracy=np.zeros(K_Max+1)
    F1_Score=np.zeros(K_Max+1)

    for k in range(2,K_Max+1):
        Y_pred=np.zeros(len(X_val))
        tree = create_decision_treeID3(X_train, min_instance_count=2,Max_Depth=k, dep
        for i in range(0,len(X_val)):
            Y_pred[i]=make_decision(X_val[i],tree)

        Accuracy[k], F1_Score[k]=CM(Y_pred, Y_val)

        print("The value of Max-Depth is %d ." %(k))
        print(F1_Score[k])
```

```
    return Accuracy,F1_Score


file = open('winequality-white.csv')

data=[]
TsAc=[]
TsF1=[]
VAc=[]
VF1=[]
TrAc=[]
TrF1=[]

for row in file:
    a=row.split(';')
    data.append(a)

del data[0]

X=np.asarray(data).astype('float')
np.random.seed(5)
np.random.shuffle(X)

X_1=X[0:np.shape(X)[0]/4]
X_2=X[np.shape(X)[0]/4:2*(np.shape(X)[0]/4)]
X_3=X[2*(np.shape(X)[0]/4):3*(np.shape(X)[0]/4)]
X_4=X[3*(np.shape(X)[0]/4):np.shape(X)[0]]

test=[X_1,X_2,X_3,X_4]
tr1=np.concatenate((X_2,X_3,X_4),axis=0)
tr2=np.concatenate((X_3,X_4,X_1),axis=0)
tr3=np.concatenate((X_4,X_1,X_2),axis=0)
tr4=np.concatenate((X_1,X_2,X_3),axis=0)
train=[tr1,tr2,tr3,tr4]

K_best_final=14

print("Hyper-parameters:")
print("Best Max-Depth in Fold: %d" %(K_best_final))

for i in range(0,4):
    X_test=test[i]
```

```
    X_train=train[i]

    TsAccuracy,TsF1_Score,VAccuracy,VF1_Score,TrAccuracy,TrF1_Score=Final_Output_T
    TsAc.append(TsAccuracy)
    TsF1.append(TsF1_Score)
    VAc.append(VAccuracy)
    VF1.append(VF1_Score)
    TrAc.append(TrAccuracy)
    TrF1.append(TrF1_Score)

    print("Fold-%d:" %(i+1))
    print("Training: F1 Score: %f , Accuracy: %f" %(TrF1_Score,TrAccuracy))
    print("Validation: F1 Score: %f , Accuracy: %f" %(VF1_Score,VAccuracy))
    print("Test: F1 Score: %f , Accuracy: %f" %(TsF1_Score,TsAccuracy))


print("Average:")
print("Training: F1 Score: %f , Accuracy: %f" %(np.mean(TrF1),np.mean(TrAc)))
print("Validation: F1 Score: %f , Accuracy: %f" %(np.mean(VF1),np.mean(VAc)))
print("Test: F1 Score: %f , Accuracy: %f" %(np.mean(TsF1),np.mean(TsAc)))

end = time.time()
print("The time taken for the algorithm computation is :- %f seconds." % (end-start

#file = open('winequality-white.csv')
#
#data=[]
#Ac=[]
#F1=[]
#
#TsAc=[]
#TsF1=[]
#VAc=[]
#VF1=[]
#TrAc=[]
#TrF1=[]
#
#for row in file:
#    a=row.split(';')
#    data.append(a)
#
#del data[0]
#
```

```
#X=np.asarray(data).astype('float')
##np.random.seed(0)
#np.random.shuffle(X)
#
#X_1=X[0:np.shape(X)[0]/4]
#X_2=X[np.shape(X)[0]/4:2*(np.shape(X)[0]/4)]
#X_3=X[2*(np.shape(X)[0]/4):3*(np.shape(X)[0]/4)]
#X_4=X[3*(np.shape(X)[0]/4):np.shape(X)[0]]
#
#test=[X_1,X_2,X_3,X_4]
#tr1=np.concatenate((X_2,X_3,X_4),axis=0)
#tr2=np.concatenate((X_3,X_4,X_1),axis=0)
#tr3=np.concatenate((X_4,X_1,X_2),axis=0)
#tr4=np.concatenate((X_1,X_2,X_3),axis=0)
#train=[tr1,tr2,tr3,tr4]
#
#for i in range(0,4):
#    X_test=test[i]
#    X_train=train[i]
#
#    Accuracy,F1_Score=cross_validation_test(X_train,25)
#    K_best_fold=np.argmax(F1_Score)
#    print("The best value of Max-Depth is %d and fold number is %d." % (K_best_fol
#    print(Accuracy[K_best_fold])
#    print(F1_Score[K_best_fold])
#
#    x = np.arange(2,26, 1)
#    Accuracy=Accuracy[2:]
#    F1_Score=F1_Score[2:]
#    F1.append(F1_Score)
#    Ac.append(Accuracy)
#
#    plt.figure(1)
#    plt.plot(x,Accuracy, label = "fold %d" %(i+1))
#    plt.figure(2)
#    plt.plot(x,F1_Score, label = "fold %d" %(i+1))
#
#    TsAccuracy,TsF1_Score,VAccuracy,VF1_Score,TrAccuracy,TrF1_Score=Final_Output_T
#    TsAc.append(TsAccuracy)
#    TsF1.append(TsF1_Score)
#    VAc.append(VAccuracy)
#    VF1.append(VF1_Score)
#    TrAc.append(TrAccuracy)
```

```
#     TrF1.append(TrF1_Score)
#
#     print("Hyper-parameters:")
#     print("Best Max-Depth in Fold: %d" %(K_best_fold))
#
#     print("Fold-%d:" %(i+1))
#     print("Training: F1 Score: %f , Accuracy: %f" %(TrF1_Score,TrAccuracy))
#     print("Validation: F1 Score: %f , Accuracy: %f" %(VF1_Score,VAccuracy))
#     print("Test: F1 Score: %f , Accuracy: %f" %(TsF1_Score,TsAccuracy))
#
#
#print("Average:")
#print("Training: F1 Score: %f , Accuracy: %f" %(np.mean(TrF1),np.mean(TrAc)))
#print("Validation: F1 Score: %f , Accuracy: %f" %(np.mean(VF1),np.mean(VAc)))
#print("Test: F1 Score: %f , Accuracy: %f" %(np.mean(TsF1),np.mean(TsAc)))
#
#
#plt.figure(1)
#plt.xlabel('K')
## naming the y axis
#plt.ylabel('Accuracy')
## giving a title to my graph
#plt.title('Accuracy vs K')
## show a legend on the plot
#plt.legend()
## function to show the plot
#plt.savefig('Accuracy.png')
#
#plt.figure(2)
#plt.xlabel('K')
## naming the y axis
#plt.ylabel('F1 Score')
## giving a title to my graph
#plt.title('F1 Scores vs K')
## show a legend on the plot
#plt.legend()
## function to show the plot
#plt.savefig('F1_Score.png')
#
#pickle.dump(X, open( "X_data_saved.p", "wb" ) )
#pickle.dump(Ac, open( "Ac_data_saved.p", "wb" ) )
#pickle.dump(F1, open( "F1_data_saved.p", "wb" ) )
#
```

```
#
#end = time.time()
#print("The time taken for the algorithm computation is :- %f seconds." % (end-star
```

The End