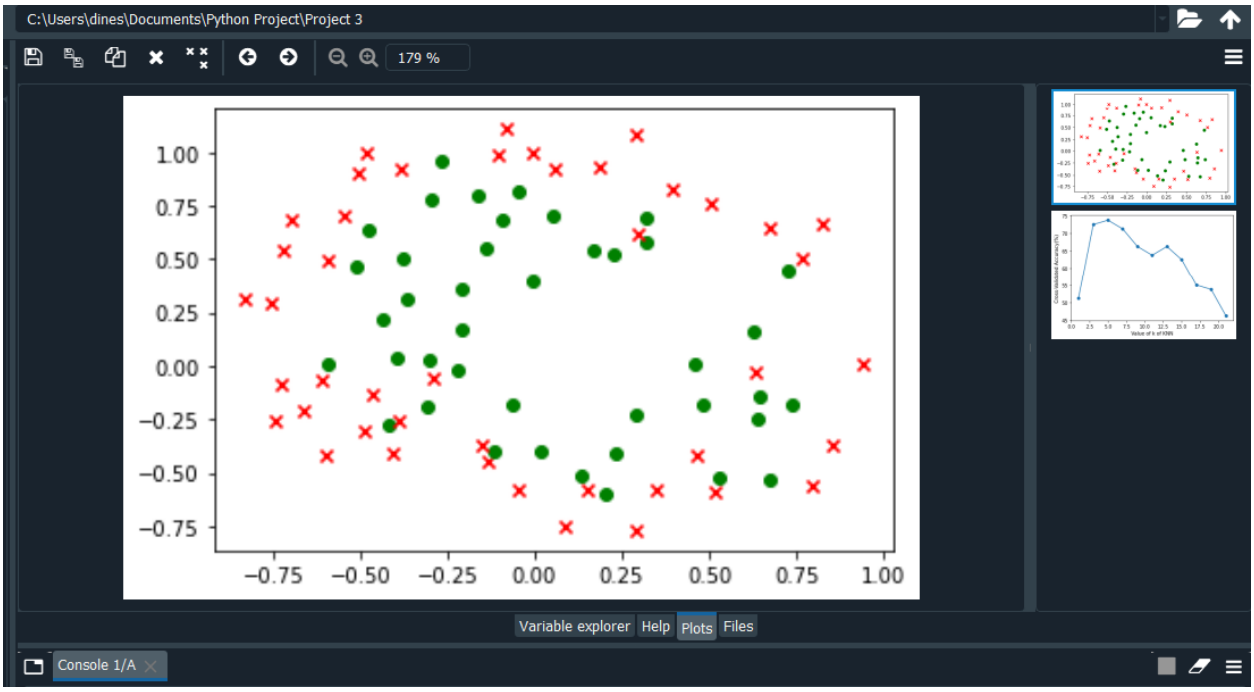
Mid-Term Project report

Title: Create a binary classifier to predict quality test results for each capacitor using kNN

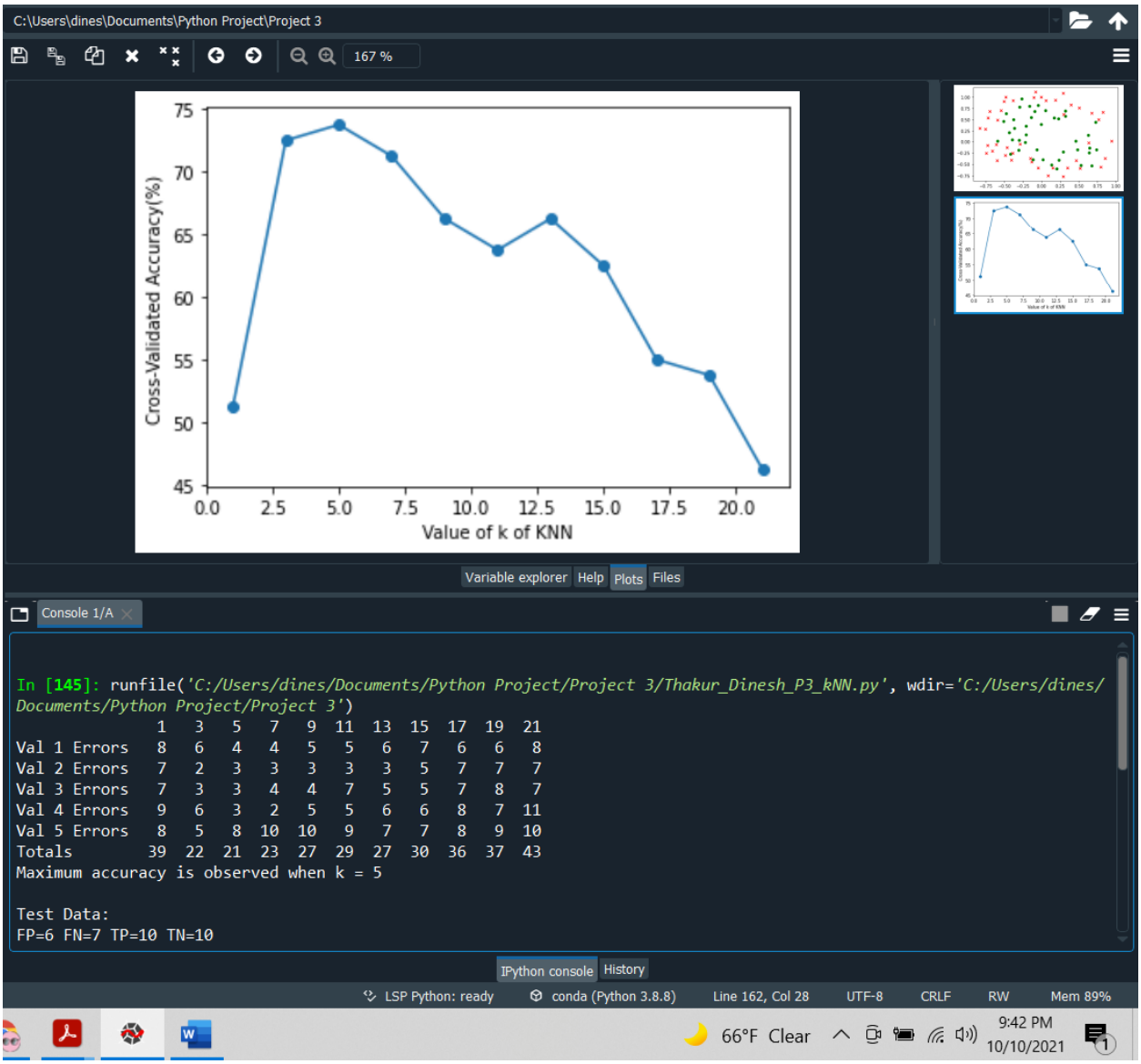
Introduction:

For this project, I have applied the kNN algorithm to predict whether capacitors from a fabrication plant pass quality control based (QC) on two different tests. The data set has a total of 118 entries that have already been randomized and separated into two datasets called P3train.txt and P3test.txt; Each tuple has a class column with either 1 or 0 value depending on whether that specific capacitor passed or failed, respectively. Below is the plot for the dataset:



Procedure:

After reading the input file from the user, the program will feed the input to the train and test data frames. For evaluating the k value, the code will split the training dataset into five-folds and use one set as a validation set and the rest as a training dataset on a rotational basis. After that, evaluate the misclassification rate for each value of k in every validation set. After comparing the average accuracy for different k values, the program will choose the best k value. Then, the k value selected and the entire training set will predict the class values for the user-given test file. Finally, the model will state the confusion matrix evaluation for its accuracy, precision, recall, and F1 values.



Predicted Class

	0	1
0	**True Negatives (TN = 10)**	**False Positives (FP = 6)**
1	**False Negatives (FN = 7)**	**True Positives (TP = 10)**

Final Results:

After training the model on k-fold training datasets, k=5 showed the maximum accuracy. So we classified the resulting pass/fail class by taking this value and predicting the test dataset based on 5 of the nearest neighbors in the training set. We classified the resulting pass/fail class. The model showed 60% accuracy, 62% precision, 58% recall value and 60% F1 value.

Code:

```
import numpy as np

import pandas as pd

from math import sqrt

import matplotlib.pyplot as plt


aFileName = input("Enter the name of your train file: ")

fin= open(aFileName, "r")

# fin= open('P3train.txt', "r")


aString = fin.readline()

d=aString.split("\t")

rows = int(d[0])

columns = int(d[1])+1


data= np.zeros([rows, columns])


for k in range(rows):

    aString = fin.readline()

    t = aString.split("\t")

    t[-1]=t[-1].strip()


    for j in range(columns):
```

```
        data[k,j]= float(t[j])

fin.close()
for i in range(rows):
    df = pd.DataFrame(data, columns =['x','y','Class'])
    if(df.values[i][-1]==1):
        plt.scatter(df.values[i][0] ,df.values[i][1] , marker='o', color="green");
    if(df.values[i][-1]==0):
        plt.scatter(df.values[i][0] ,df.values[i][1] , marker='x', color="red");
plt.show()
# print(df)



k=rows//5
#k-fold implementation:
k1 = df.iloc[0:k,:]
k2 = df.iloc[k:2*k,:]
k3 = df.iloc[2*k:3*k,:]
k4 = df.iloc[3*k:4*k,:]
k5 = df.iloc[4*k:5*k,:]

# print(k1)
# print(k2)
# print(k3)
# print(k4)
# print(k5)
TrainingSet=np.concatenate((k1,k2,k3,k4,k5))
```

```
ValidationSet1=k1
TrainingSet1=np.concatenate((k2,k3,k4,k5))

ValidationSet2=k2
TrainingSet2=np.concatenate((k1,k3,k4,k5))

ValidationSet3=k3
TrainingSet3=np.concatenate((k1,k2,k4,k5))

ValidationSet4=k4
TrainingSet4=np.concatenate((k1,k2,k3,k5))

ValidationSet5=k5
TrainingSet5=np.concatenate((k1,k2,k3,k4))

def e_Dist(coordinate_1, coordinate_2):
        distance = 0.0
        for i in range(len(coordinate_1)-1):
                distance += (coordinate_1[i] - coordinate_2[i])**2
        return sqrt(distance)



# Locate the nearest neighbors
def nearest_pt(train, test_Pt, k_neighbors):
        distances = list()
        for current_row in train:
                dist = e_Dist(test_Pt, current_row)
                distances.append((current_row, dist))
```

```
        distances.sort(key=lambda tup: tup[1])
        Nearest_N = list()
        for i in range(k_neighbors):
                Nearest_N.append(distances[i][0])
        return Nearest_N

# Make a classification prediction with neighbors
def predict_Class(train, test_row, k_neighbors):
        Nearest_N = nearest_pt(train, test_row, k_neighbors)
        output_values = [row[-1] for row in Nearest_N]
        prediction = max(set(output_values), key=output_values.count)
        return prediction



#err_List = list()
# kFold=[]
k1Fold=[]
k2Fold=[]
k3Fold=[]
k4Fold=[]
k5Fold=[]
for k_value in range(1,23,2):
   errCount=0
   for i in range(k):
      prediction = predict_Class(TrainingSet1, ValidationSet1.values[i], k_value)
      if(prediction!=ValidationSet1.values[i][-1]):
         errCount+=1
      #print('%d Expected class %d, Predicted class %d.' % (i,ValidationSet1.values[i][-1],
prediction))
```

```
    k1Fold.append(errCount)
    #err_List.append(errCount)
    #print("Total Errors for k=%d are %d" %(k_value, errCount))

#kFold = pd.DataFrame(err_List).T

for k_value in range(1,23,2):
    errCount=0
    for i in range(k):
        prediction = predict_Class(TrainingSet2, ValidationSet2.values[i], k_value)
        if(prediction!=ValidationSet2.values[i][-1]):
            errCount+=1
            #print('%d Expected class %d, Predicted class %d.' % (i,ValidationSet1.values[i][-1],
prediction))
    k2Fold.append(errCount)
    #print("Total Errors for k=%d are %d" %(k_value, errCount))



for k_value in range(1,23,2):
    errCount=0
    for i in range(k):
        prediction = predict_Class(TrainingSet3, ValidationSet3.values[i], k_value)
        if(prediction!=ValidationSet3.values[i][-1]):
            errCount+=1
            #print('%d Expected class %d, Predicted class %d.' % (i,ValidationSet1.values[i][-1],
prediction))

    k3Fold.append(errCount)
```

```
    #print("Total Errors for k=%d are %d" %(k_value, errCount))


for k_value in range(1,23,2):
    errCount=0
    for i in range(k):
        prediction = predict_Class(TrainingSet4, ValidationSet4.values[i], k_value)
        if(prediction!=ValidationSet4.values[i][-1]):
            errCount+=1
            #print('%d Expected class %d, Predicted class %d.' % (i,ValidationSet1.values[i][-1],
prediction))


    k4Fold.append(errCount)
    #print("Total Errors for k=%d are %d" %(k_value, errCount))


for k_value in range(1,23,2):
    errCount=0
    for i in range(k):
        prediction = predict_Class(TrainingSet5, ValidationSet5.values[i], k_value)
        if(prediction!=ValidationSet5.values[i][-1]):
            errCount+=1
            #print('%d Expected class %d, Predicted class %d.' % (i,ValidationSet1.values[i][-1],
prediction))


    k5Fold.append(errCount)
    #print("Total Errors for k=%d are %d" %(k_value, errCount))
data = [k1Fold, k2Fold, k3Fold, k4Fold, k5Fold]
kFold = pd.DataFrame(data, columns=['1','3','5','7','9','11','13','15','17','19','21'])
kFold= kFold.append(kFold.sum(axis=0, skipna=True),ignore_index=True)
kFold.index = ['Val 1 Errors','Val 2 Errors','Val 3 Errors','Val 4 Errors','Val 5 Errors','Totals']
```

```
#kFold.append(kFold.sum(axis=0),ignore_index=True)
print(kFold)

Totals = kFold.iloc[-1,:].values.tolist()
#Calculation of Average Accuracy

Accuracy=[]

for z in range(len(Totals)):
    Accuracy.append((1-(Totals[z]/(16*5)))*100)
    # print('Average Accuracy = %f percent' % Acc_k)
#print(Accuracy)

#Plotting the graph
kValues=[1,3,5,7,9,11,13,15,17,19,21]
plt.plot(kValues, Accuracy, marker = 'o')
plt.xlabel("Value of k of KNN")
plt.ylabel("Cross-Validated Accuracy(%)")
plt.show()


max_value = max(Accuracy)
max_index = Accuracy.index(max_value)
# print(max_index)
print("Maximum accuracy is observed when k =", kFold.columns[max_index])
k_Final = int(kFold.columns[max_index])

# Testing our model on User Given data
```

```
aFileName = input("Enter the name of your test file: ")
fin= open(aFileName, "r")
# fin= open('P3test.txt', "r")

aString = fin.readline()
d=aString.split("\t")
rows1 = int(d[0])
columns1 = int(d[1])+1

data= np.zeros([rows1, columns1])

for i in range(rows1):
    aString = fin.readline()
    t = aString.split("\t")
    t[-1]=t[-1].strip()

    for j in range(columns1):
        data[i,j]= float(t[j])

fin.close()

td = pd.DataFrame(data, columns =['x','y','Class'])
# print(dt)

TD=td.iloc[:,:]
# print(TD)
# print(TrainingSet1)
```

```
FP=FN=TP=TN=0
for i in range(rows1):
    F_prediction = predict_Class(TrainingSet, TD.values[i], k_Final)
    if(TD.values[i][-1]==0 and F_prediction==1):
        FP+=1
    if(TD.values[i][-1]==1 and F_prediction==0):
        FN+=1
    if(TD.values[i][-1]==1 and F_prediction==1):
        TP+=1
    if(TD.values[i][-1]==0 and F_prediction==0):
        TN+=1
    # print('Expected %d, Got %d.' % (TD.values[i][-1], F_prediction))
print('\nTest Data: \nFP=%d FN=%d TP=%d TN=%d'%(FP, FN, TP, TN))
Model_Acc=((TP+TN)/(TP+TN+FP+FN))*100
Model_Prec=(TP/(TP+FP))*100
Recall=(TP/(TP+FN))*100
F1=2*(1/((1/Model_Prec)+(1/Recall)))
print('Accuracy for test set = %d percent' %Model_Acc)
print('Precision for test set = %d percent' %Model_Prec)
print('Recall for test set = %d percent' %Recall)
print('F1 for test set = %d percent' %F1)
```