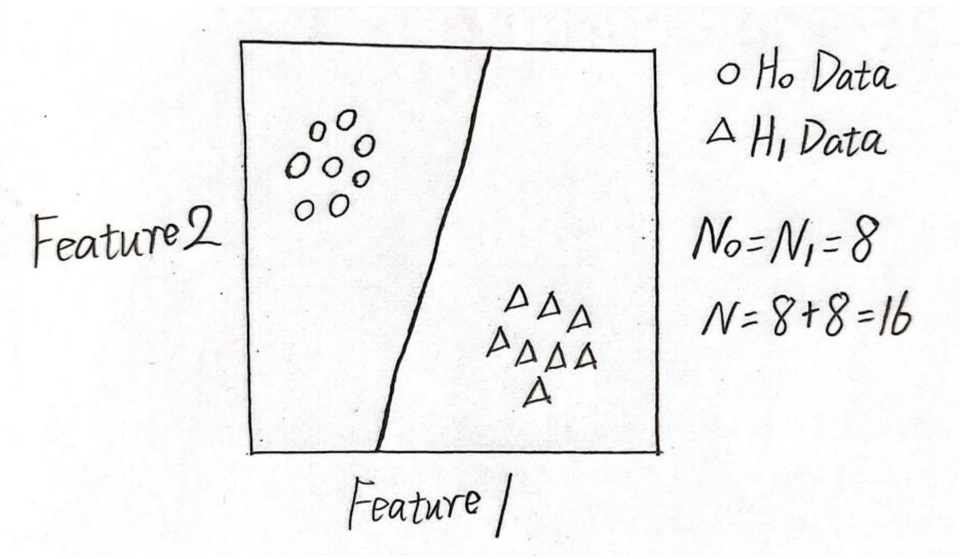
ECE 580 Homework 4

Libo Zhang (lz200)

Exploring K-Nearest Neighbors Section

Question 1 (a) Solution:



In the above sketch example, assume we have N0 = N1 = 8 (actually, this number can be extended to a very large value, as long as the H0 data cluster and H1 data cluster are far away from each other), and we can easily know that for any value of k <= N/2 = 8, the KNN classifier can achieve the perfect performance when being tested on the training data.

Question 1 (b) Solution:

If the classes are not balanced, first we should find which class has a smaller number of training points, denoted as N_{min}. Then we should consider the boundary condition: Suppose H0 has less training points, for a large k, if we have included all N_{min} for one H0 training/testing point, what is the maximum number of H1 training/testing point we can include so that the KNN classifier can still correctly predict H0? Clearly, we can include at most another $(N_{min} - 1)$ H1 points, and the KNN classifier can still correctly predict H0. Therefore, the maximum value of k should be determined by $k_{max} = 2 \times \min(N_0, N_1) - 1$. Then, for any value of $k \leq k_{max}$, we can still achieve the perfect performance when the KNN classifier is tested on the imbalanced training data.

Question 1 (c) Solution:

To begin, let us consider two boundary conditions.

Firstly, when $N_0 = 1, N_1 = N - 1$, if we want the KNN classifier to correctly classify the only H0 training point, we have to set k = 1. Otherwise (for all other values of k), the KNN classification decisions will always be H1 based on the majority vote decision boundary.

Secondly, when $N_0 = N - 1, N_1 = 1$, if we want the KNN classifier to correctly classify the only H1 training point, we need to set k = 1 or k = 2, based on the majority vote decision boundary (k = 2 is OK here because H1 is our class of interest, so when the decision statistic is greater than **or equal to** 0.5, we choose H1). Otherwise (for all other values of k), the KNN classification decisions will always go to H0 based on the majority vote decision boundary.

To summarize, if the classes in the training data are imbalanced, based on the majority vote decision boundary, the KNN classification decisions tend to more frequently, go to the classes which have more training points. This will have negative effects on the probability of correct decision.

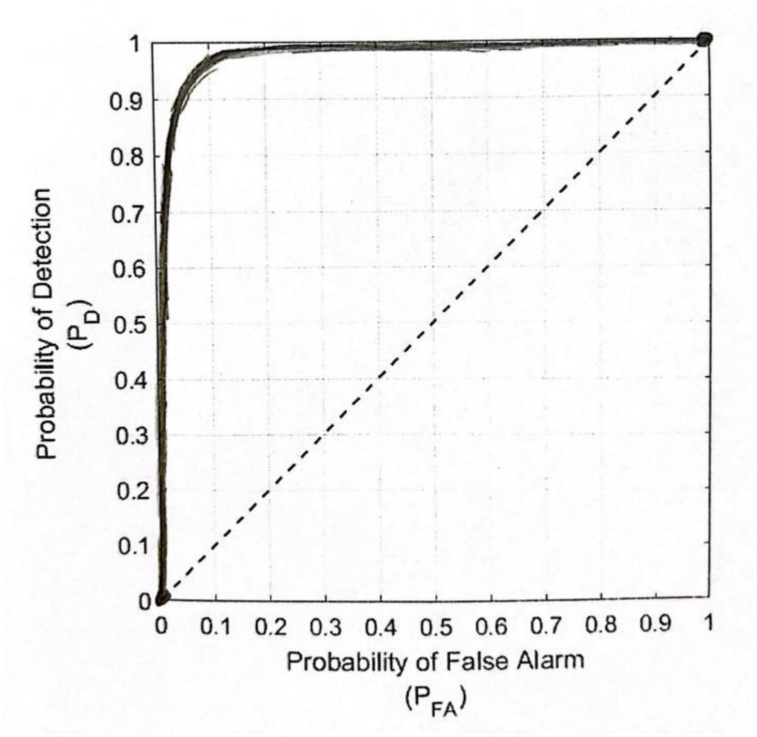
Question 1 (d) Solution:

If the environment in which my KNN classifier will be deployed has balanced classes, but the classes in my training data are imbalanced, I will use Data Scaling (Normalization) to normalize all data to a similar scale before performing classification. I hope that modifying my KNN classifier by data scaling (normalization) can change the decision statistics and final KNN classification decisions, which eventually might help improve classification. It should be noted that if we scale or normalize the imbalanced training data, when the KNN classifier is deployed in the environment of balanced classes, we should also scale or normalize the balanced testing data using the same data scaling method.

Exploring Cross-Validation within the Context of Classification Section

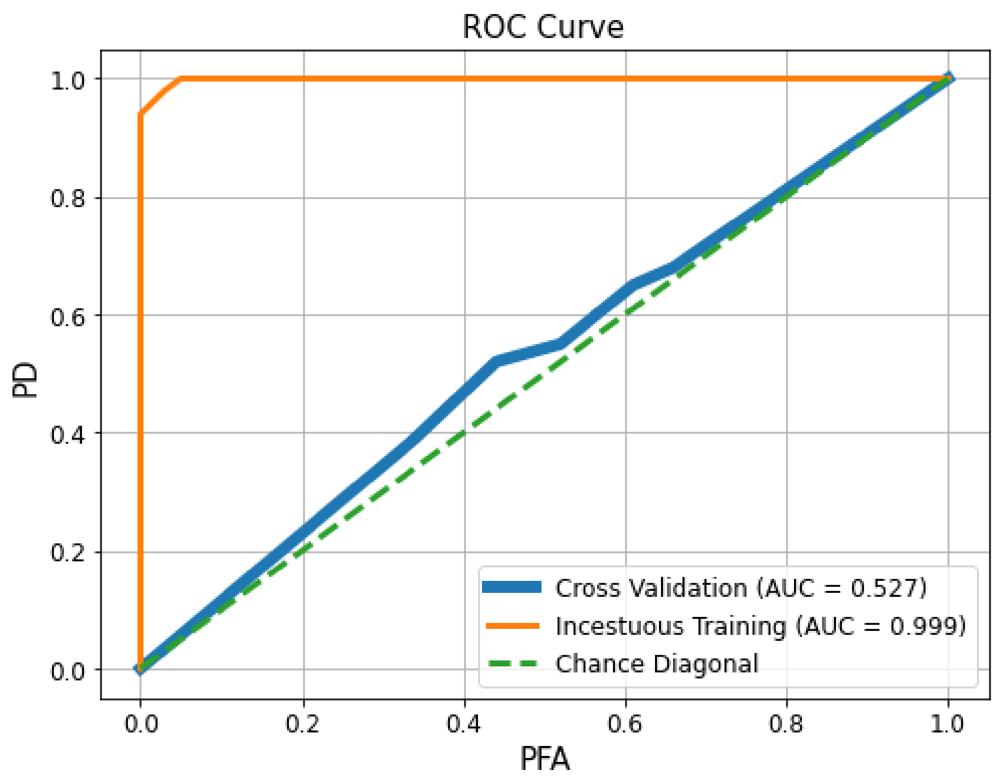
Question 2 (a) Solution:

From visual inspection of this dataset, I think the binary classification task for a KNN classifier with k = 5 will be relatively easy, because data points of class 0 and class 1 are visually separable. Therefore, I think the ROC curve should be very close to the top-left corner, and my qualitatively sketch is shown below.



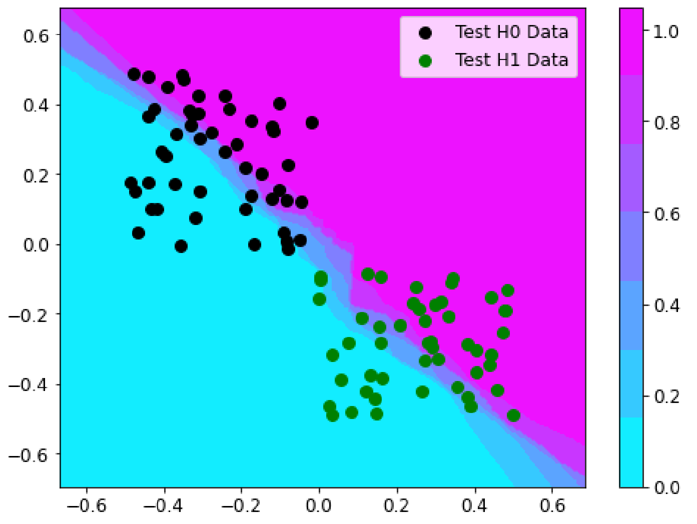
Question 2 (b) Solution:

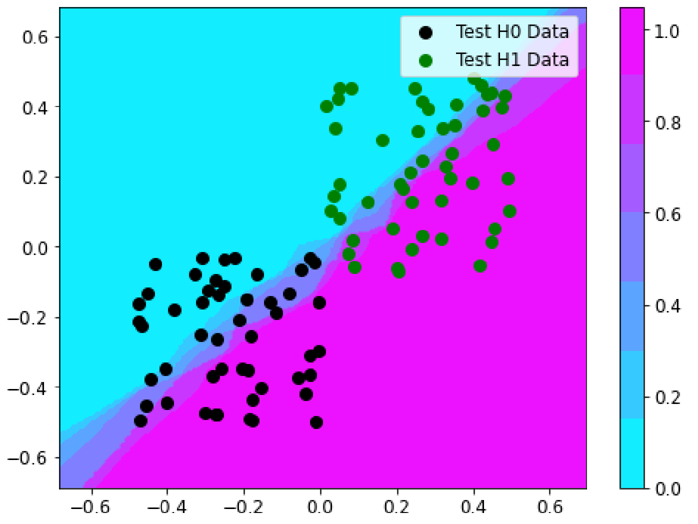
Question 2 (b) (i) – The cross-validated ROC and incestuous training ROC when applying a KNN classifier with k = 5, are shown below (2 ROCs are plotted within the same figure axis for better comparison).



Question 2 (b) Solution:

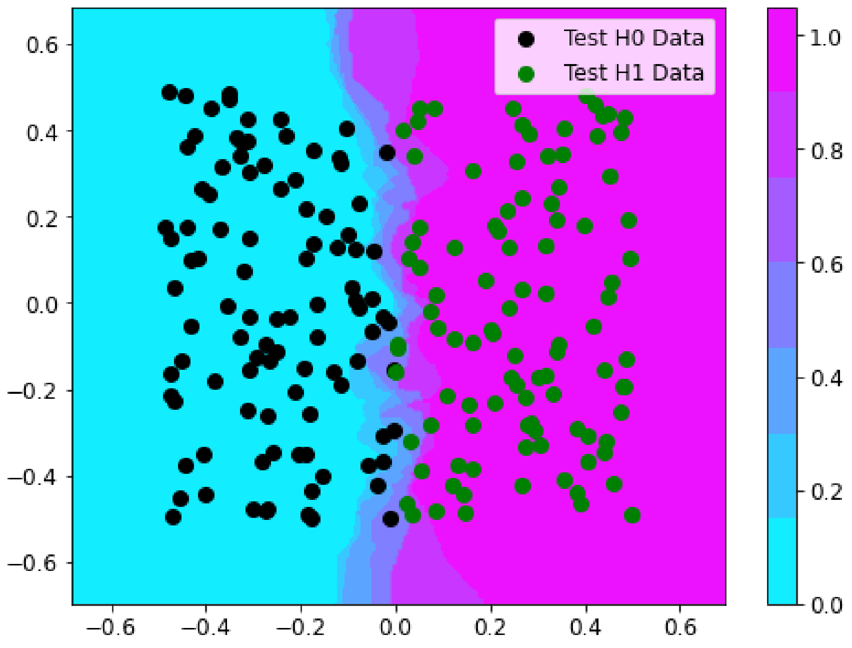
Question 2 (b) (ii) – We can see that the incestuous training ROC is very close to the "perfect" top-left corner and has very large area under curve (AUC), which is great. However, the cross-validated ROC is very close to the chance diagonal, meaning that the cross-validated KNN classifier is almost randomly guessing the binary classification, which is very bad. To explain, first I will plot two decision statistic surfaces for the 2 folds below.





Clearly, there are some problems when assigning data points to these 2 cross-validated folds. From visual inspection we should realize that the ideal decision boundary should look like a vertical line separating two classes in the middle. However, here the data points are assigned to each fold diagonally, which explains why the cross-validated ROC is very close to the random guessing chance diagonal.

To further demonstrate my explanation, the decision statistic surface of incestuous training is shown below.



Note that for incestuous training, the testing data points are the training data points. Comparing the incestuous training decision statistic surface with the decision statistic surfaces of cross validation, we can conclude that my explanation is indeed reasonable.

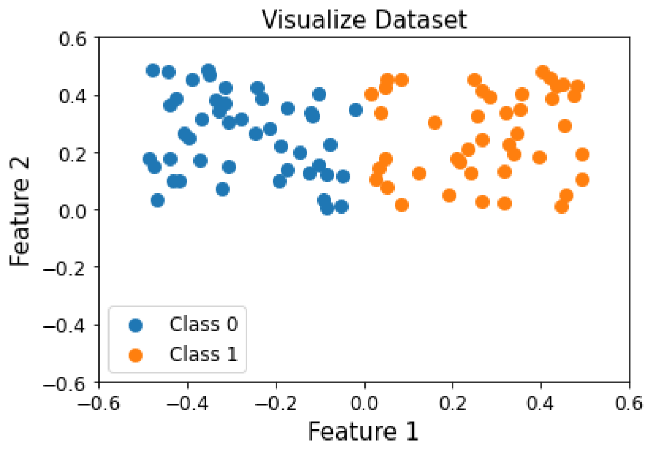
Question 2 (c) Solution:

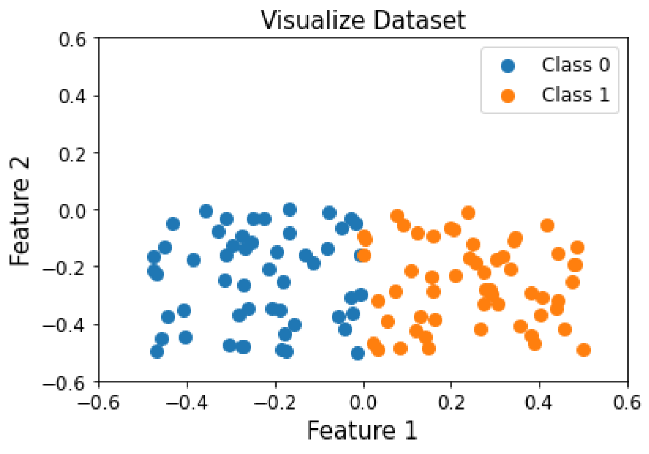
To modify the cross-validation, I think we should assign data points to each fold based on the values of the second feature (Feature 2), and the specific assigning method is described below.

Firstly, assign all data points (both class 0 and 1) which have non-negative Feature 2 values to Fold 1.

Secondly, assign all data points (both class 0 and 1) which have negative Feature 2 values to Fold 2.

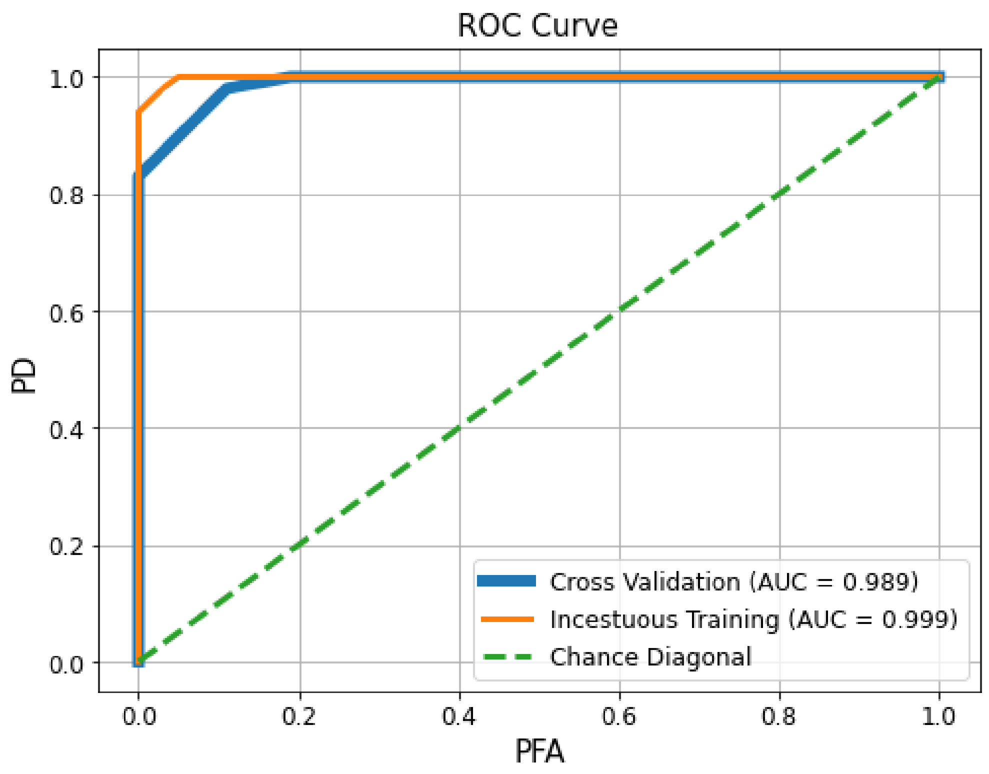
Finally, let us visualize these 2 folds for new cross-validation below (left for Fold 1, right for Fold 2).





Question 2 (d) Solution:

The modified cross-validation ROC and the incestuous ROC are shown below.



We can see that both the modified cross-validation ROC and the incestuous training ROC are close to each other and close to the top-left corner. Both ROCs have very high AUC values. Therefore, we have successfully modified the cross-validation to provide more reasonable representative estimates of classifier performance for a KNN classifier with k = 5.

Question 2 (e) Solution:

Reference the Packages/Toolboxes

I use numpy to process arraies and matrices.
I use pandas to read and process the original datasets.
I use matplotlib.pyplot to plot ROC curves and decision statistic surfaces.
To plot decision statistic surfaces, I also referenced the ECE580 course materials and the website below.
https://hackernoon.com/how-to-plot-a-decision-boundary-for-machine-learning-algorithms-in-python-3o1n3w07
I use metrics from sklearn to calculate area under curve (AUC).
I use sklearn to build and implement KNN classifiers.

---

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
```

[2]:
```
filename = "dataSetCrossValWithKeys.csv"
dataframe = pd.read_csv(filename, header = None)
```

[3]:
```
data = dataframe.to_numpy()
X_data = data[:, 2:]
y_data = data[:, 1]

print(data.shape)
print(data[0:3])
```

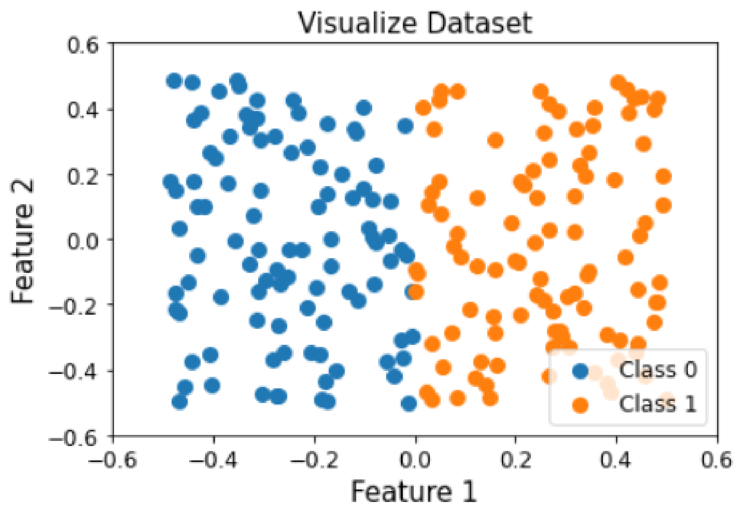
```
(200, 4)
[[ 2.         1.         0.48512 -0.13156]
 [ 1.         1.         0.44915  0.43686]
 [ 2.         0.        -0.10335  0.40376]]
```

[4]:
```
fold1 = data[data[:,0] == 1., :]
fold2 = data[data[:,0] == 2., :]
X_fold1 = fold1[:, 2:]
X_fold2 = fold2[:, 2:]
y_fold1 = fold1[:, 1]
y_fold2 = fold2[:, 1]

print(fold1.shape)
print(fold2.shape)
print(X_fold1.shape)
print(X_fold2.shape)
print(y_fold1.shape)
print(y_fold2.shape)
```

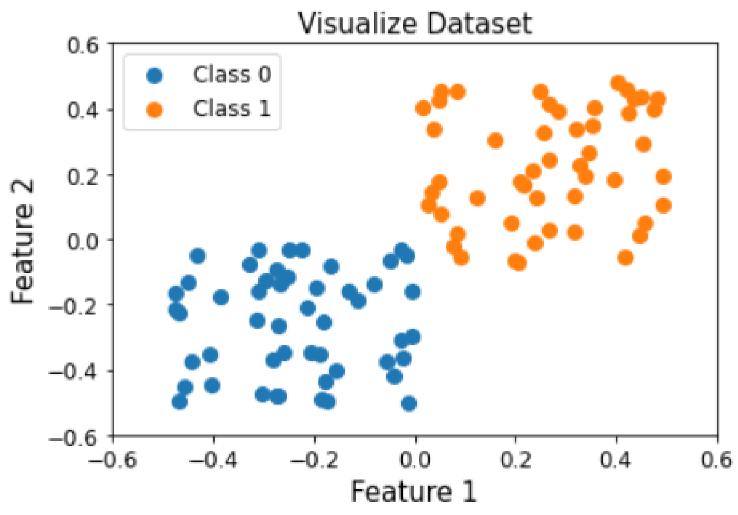
```
(100, 4)
(100, 4)
(100, 2)
(100, 2)
(100,)
(100,)
```

```
[5]: def visualize_dataset (data) :
         class0 = data[data[:,1] == 0., 2:]
         class1 = data[data[:,1] == 1., 2:]
         figure, axis = plt.subplots()
         axis.scatter(class0[:, 0], class0[:, 1], label = "Class 0", linewidths = 3)
         axis.scatter(class1[:, 0], class1[:, 1], label = "Class 1", linewidths = 3)
         axis.set_xlabel("Feature 1", fontsize = 15)
         axis.set_ylabel("Feature 2", fontsize = 15)
         axis.set_title("Visualize Dataset", fontsize = 15)
         #axis.grid()
         figure.set_size_inches(6, 4)
         plt.xticks(np.linspace(-0.6, 0.6, num = 7), fontsize = 12)
         plt.yticks(np.linspace(-0.6, 0.6, num = 7), fontsize = 12)
         plt.legend(fontsize = 12)
         plt.show()
         return None
```

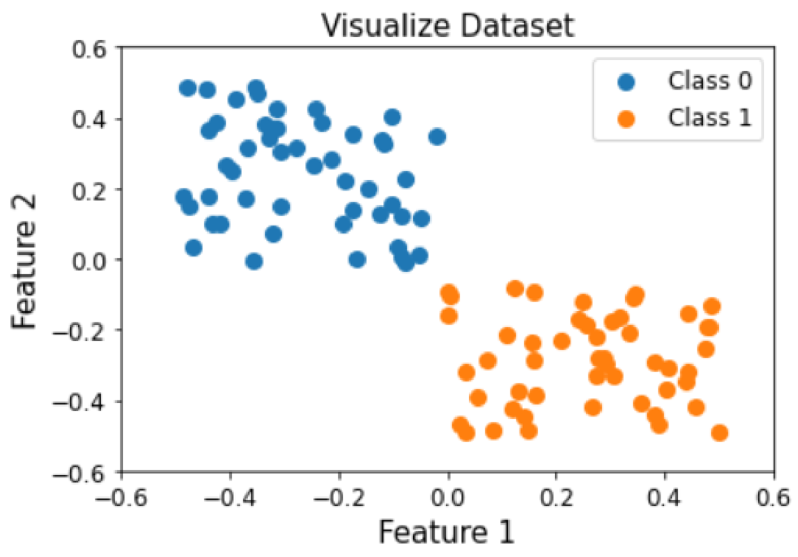
```
[6]: visualize_dataset(data)
```



```
[7]: visualize_dataset(fold1)
```



```
[8]: visualize_dataset(fold2)
```



```
[9]: knnthres = np.array([0.0, 0.2, 0.4, 0.6, 0.8, 1.0, float("inf")])
     print(knnthres.shape)

     (7,)
```

```
[10]: model_cv1 = KNeighborsClassifier(n_neighbors = 5)
      model_cv2 = KNeighborsClassifier(n_neighbors = 5)
      model_all = KNeighborsClassifier(n_neighbors = 5)
```

```
In  [11]: model_cv1.fit(X_fold1, y_fold1)

Out[11]: KNeighborsClassifier()
```

```
In  [12]: model_cv2.fit(X_fold2, y_fold2)

Out[12]: KNeighborsClassifier()
```

```
In  [13]: model_all.fit(X_data, y_data)

Out[13]: KNeighborsClassifier()
```

```
In  [14]: prob_cv1 = model_cv1.predict_proba(X_fold2)
          prob_cv2 = model_cv2.predict_proba(X_fold1)
          prob_all = model_all.predict_proba(X_data)
          print(prob_cv1.shape)
          print(prob_cv2.shape)
          print(prob_all.shape)
          print(model_cv1.score(X_fold2, y_fold2))
          print(model_cv2.score(X_fold1, y_fold1))
          print(model_all.score(X_data, y_data))

          (100, 2)
          (100, 2)
          (200, 2)
          0.54
          0.49
          0.975
```

[15]:
```
decsta_cv = np.vstack((prob_cv2, prob_cv1))
labels_cv = np.concatenate((y_fold1, y_fold2))
print(decsta_cv.shape)
print(labels_cv.shape)
```

```
(200, 2)
(200,)
```

[16]:
```
arrdata_cv = np.vstack((labels_cv, decsta_cv[:, 1])).T
arrdata_all = np.vstack((y_data, prob_all[:, 1])).T
print(arrdata_cv.shape)
print(arrdata_all.shape)
```

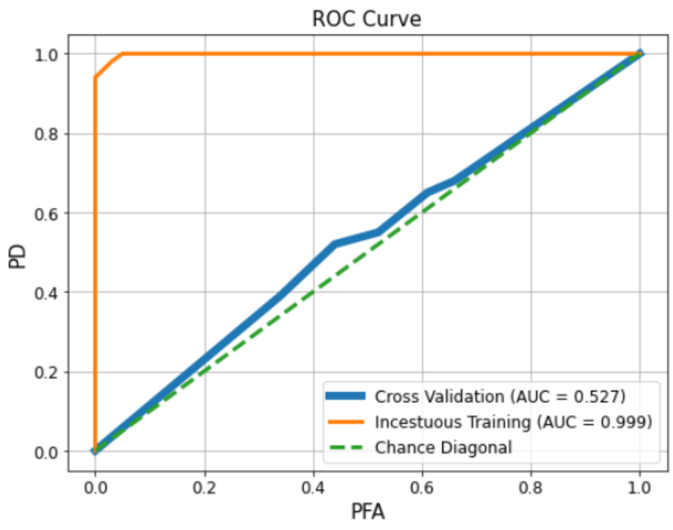
```
(200, 2)
(200, 2)
```

[17]:
```
def divide_labels (data) :
    npdata = np.copy(data)
    data0 = npdata[npdata[:,0] == 0., :]
    data1 = npdata[npdata[:,0] == 1., :]
    return data0, data1

def compute_PFA (H0, thres) :
    H0_ds = np.sort(H0[:, 1])
    false_alarm = float(len(H0_ds[H0_ds >= thres])) / (H0.shape[0])
    return false_alarm

def compute_PD (H1, thres) :
    H1_ds = np.sort(H1[:, 1])
    detection = float(len(H1_ds[H1_ds >= thres])) / (H1.shape[0])
    return detection
```

[18]:
```
def compute_ROC (arr_data, knnthres) :
    arr_PD = np.zeros(len(knnthres))
    arr_PFA = np.zeros(len(knnthres))
    H0, H1 = divide_labels(arr_data)
    for i in range(len(knnthres)) :
        arr_PD[i] = compute_PD(H1, knnthres[i])
        arr_PFA[i] = compute_PFA(H0, knnthres[i])
    return np.flipud(arr_PD), np.flipud(arr_PFA)
```

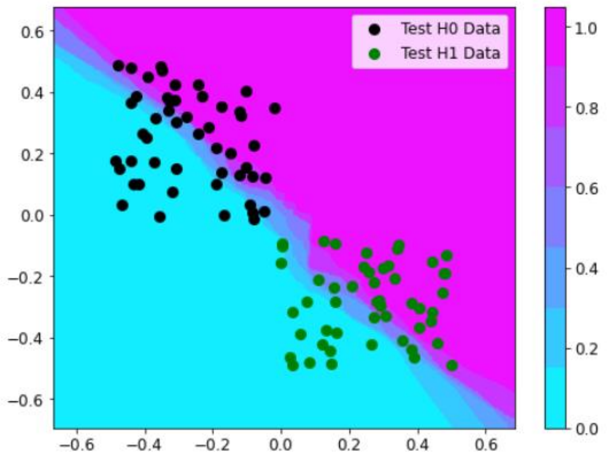
[19]:
```
def plot_ROC (arr_data_cv, arr_data_all, knnthres) :
    s1_pd, s1_pfa = compute_ROC(arr_data_cv, knnthres)
    s2_pd, s2_pfa = compute_ROC(arr_data_all, knnthres)
    auc1 = metrics.auc(s1_pfa, s1_pd)
    auc2 = metrics.auc(s2_pfa, s2_pd)
    refln = np.linspace(0, 1, num = 101)
    figure, axis = plt.subplots()
    axis.plot(s1_pfa, s1_pd, label = "Cross Validation (AUC = %.3f)" % auc1, linewidth = 6)
    axis.plot(s2_pfa, s2_pd, label = "Incestuous Training (AUC = %.3f)" % auc2, linewidth = 3)
    axis.plot(refln, refln, "--", label = "Chance Diagonal", linewidth = 3)
    axis.set_xlabel("PFA", fontsize = 15)
    axis.set_ylabel("PD", fontsize = 15)
    axis.set_title("ROC Curve", fontsize = 15)
    axis.grid()
    figure.set_size_inches(8, 6)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.legend(fontsize = 12)
    plt.show()
    return None
```

[20]:
```
plot_ROC(arrdata_cv, arrdata_all, knnthres)
```

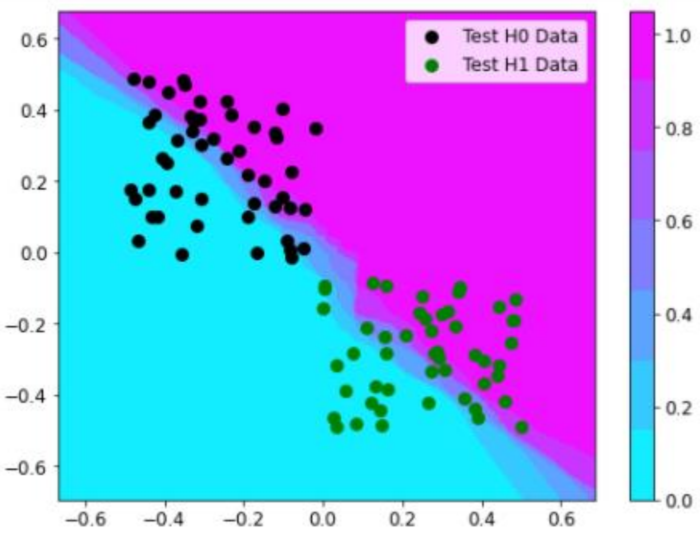


[21]:
```
def decision_statistic_surface (xTrain, xTest, model) :
    x1max = np.max(xTrain[:, 2])
    x1min = np.min(xTrain[:, 2])
    x2max = np.max(xTrain[:, 3])
    x2min = np.min(xTrain[:, 3])
    x1Range = x1max - x1min
    x2Range = x2max - x2min
    x1 = np.linspace((x1min - (0.2 * x1Range)), (x1max + (0.2 * x1Range)), 251)
    x2 = np.linspace((x2min - (0.2 * x2Range)), (x2max + (0.2 * x2Range)), 251)
    xx, yy = np.meshgrid(x1, x2)
    r1, r2 = xx.flatten(), yy.flatten()
    r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
    grid = np.hstack((r1, r2))
    yhat = model.predict_proba(grid)
    yhat_H1 = yhat[:, 1]
    zz = yhat_H1.reshape(xx.shape)
    figure, axis = plt.subplots()
    c = axis.contourf(xx, yy, zz, cmap = mpl.cm.cool)
    bounds = np.array([0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
    adjust = plt.colorbar(c, ticks = bounds)
    adjust.ax.tick_params(labelsize = 12)
    #axis.scatter(xTrain[xTrain[:, 1]==0., 2], xTrain[xTrain[:, 1]==0., 3], label = "Train H0", linewidths = 3, c = "y")
    #axis.scatter(xTrain[xTrain[:, 1]==1., 2], xTrain[xTrain[:, 1]==1., 3], label = "Train H1", linewidths = 3, c = "r")
    axis.scatter(xTest[xTest[:,1]==0., 2], xTest[xTest[:,1]==0., 3], label = "Test H0 Data", linewidths = 3, c = "k")
    axis.scatter(xTest[xTest[:,1]==1., 2], xTest[xTest[:,1]==1., 3], label = "Test H1 Data", linewidths = 3, c = "g")
    figure.set_size_inches((8, 6))
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.legend(fontsize = 12)
    plt.show()
    return None
```

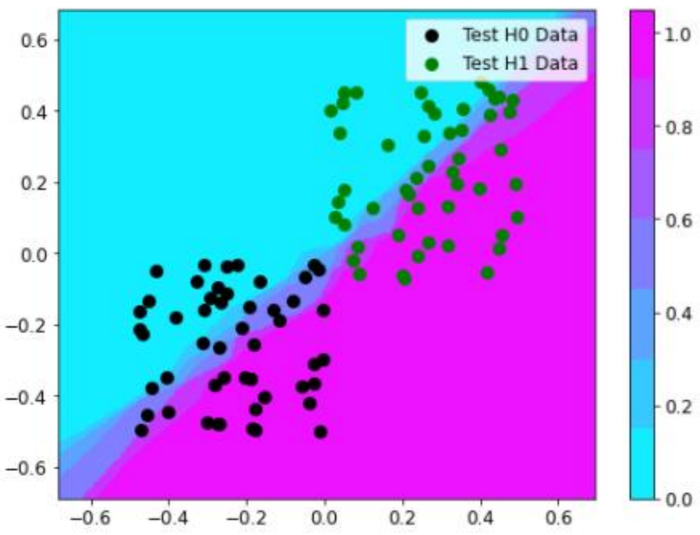
[22]:
```
decision_statistic_surface(xTrain = fold1, xTest = fold2, model = model_cv1)
```



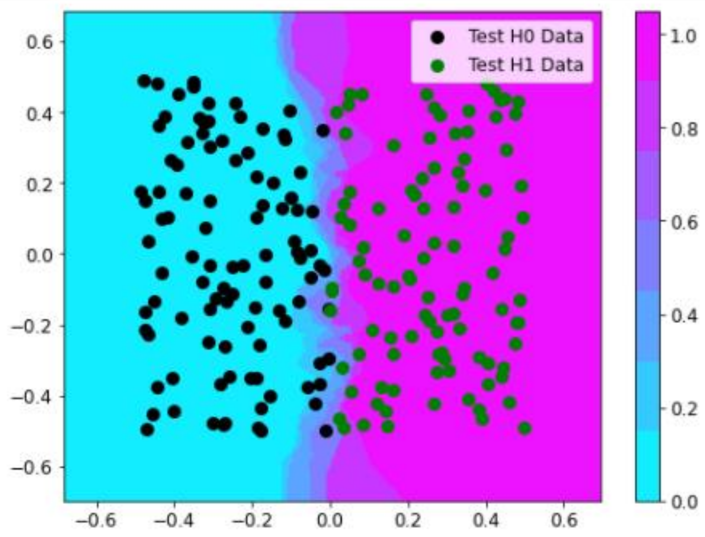
[22]:
```
decision_statistic_surface(xTrain = fold1, xTest = fold2, model = model_cv1)
```



[23]:
```
decision_statistic_surface(xTrain = fold2, xTest = fold1, model = model_cv2)
```



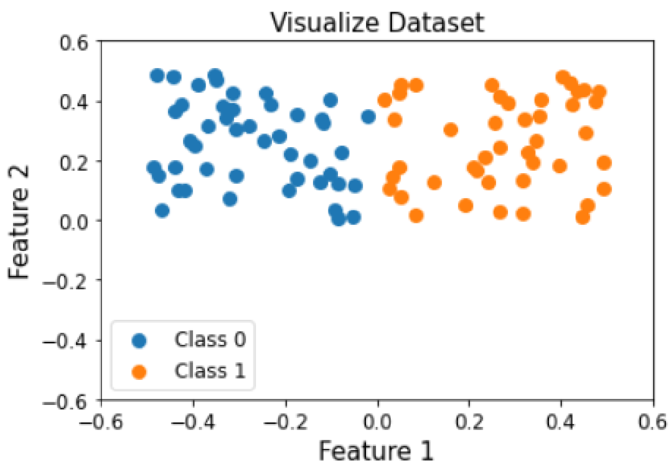
[24]:
```
decision_statistic_surface(xTrain = data, xTest = data, model = model_all)
```



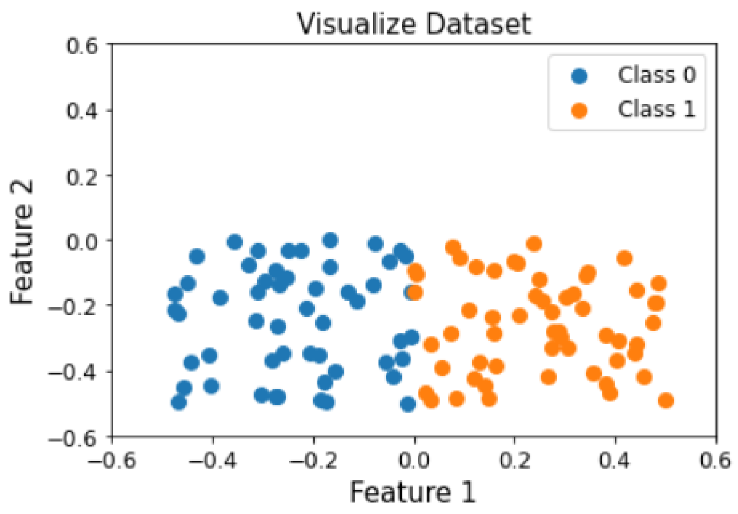
[26]:
```
fold1_new = data[data[:, 3] >= 0.0, :]
fold2_new = data[data[:, 3] < 0.0, :]
X_fold1_new = fold1_new[:, 2:]
X_fold2_new = fold2_new[:, 2:]
y_fold1_new = fold1_new[:, 1]
y_fold2_new = fold2_new[:, 1]

print(fold1_new.shape)
print(fold2_new.shape)
print(y_fold1_new.shape)
print(y_fold2_new.shape)
```

```
(91, 4)
(109, 4)
(91,)
(109,)
```

[27]:
```
visualize_dataset(fold1_new)
```



In [28]:
```
visualize_dataset(fold2_new)
```



In [29]:
```
model_new_cv1 = KNeighborsClassifier(n_neighbors = 5)
model_new_cv2 = KNeighborsClassifier(n_neighbors = 5)
```

In [30]:
```
model_new_cv1.fit(X_fold1_new, y_fold1_new)
```

Out[30]:
```
KNeighborsClassifier()
```

In [31]:
```
model_new_cv2.fit(X_fold2_new, y_fold2_new)
```

Out[31]:
```
KNeighborsClassifier()
```

```
[32]: prob_new_cv1 = model_new_cv1.predict_proba(X_fold2_new)
      prob_new_cv2 = model_new_cv2.predict_proba(X_fold1_new)
      prob_all = model_all.predict_proba(X_data)
      print(prob_new_cv1.shape)
      print(prob_new_cv2.shape)
      print(prob_all.shape)
      print("%.4f" % model_new_cv1.score(X_fold2_new, y_fold2_new))
      print("%.4f" % model_new_cv2.score(X_fold1_new, y_fold1_new))
      print(model_all.score(X_data, y_data))
```

```
(109, 2)
(91, 2)
(200, 2)
0.9266
0.9011
0.975
```

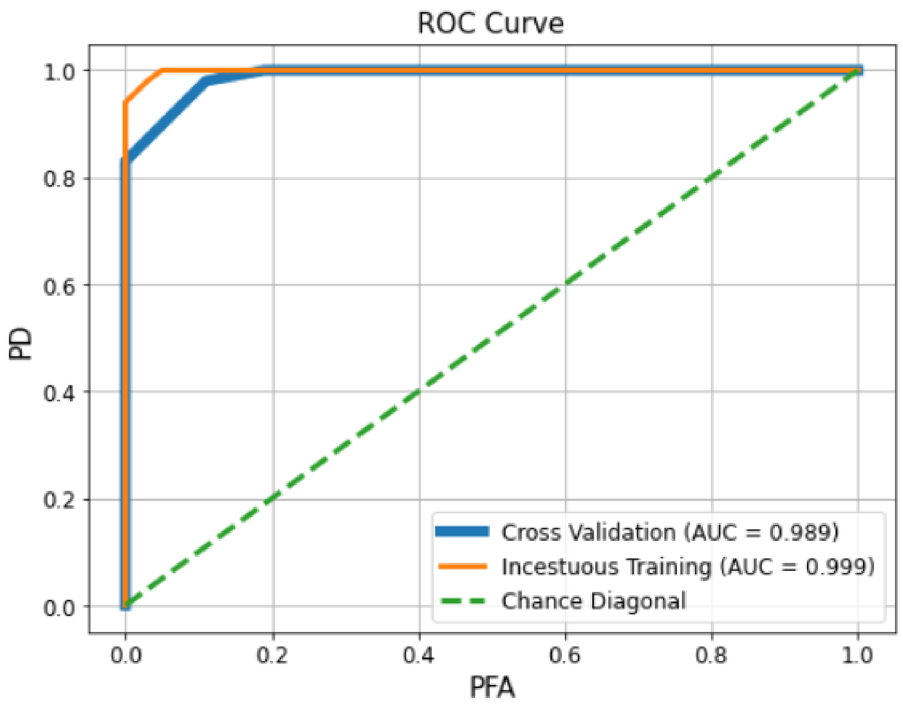
```
[33]: decsta_newcv = np.vstack((prob_new_cv2, prob_new_cv1))
      labels_newcv = np.concatenate((y_fold1_new, y_fold2_new))
      print(decsta_newcv.shape)
      print(labels_newcv.shape)
```

```
(200, 2)
(200,)
```

```
[34]: arrdata_newcv = np.vstack((labels_newcv, decsta_newcv[:, 1])).T
      arrdata_all = np.vstack((y_data, prob_all[:, 1])).T
      print(arrdata_newcv.shape)
      print(arrdata_all.shape)
```

```
(200, 2)
(200, 2)
```

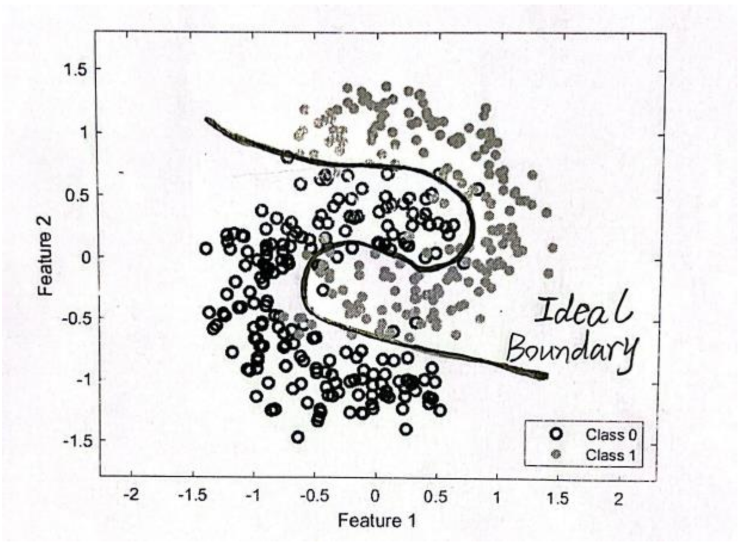
```
[35]: plot_ROC(arrdata_newcv, arrdata_all, knnthres)
```



Applying KNN to Explore Bias-Variance Trade-off in Classification Section

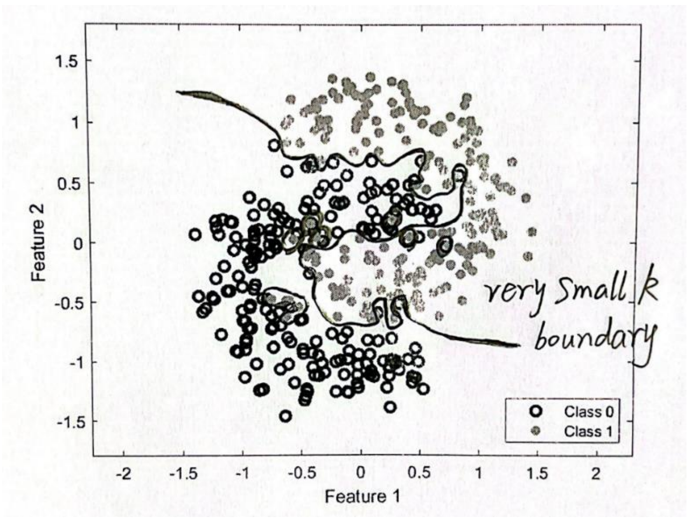
Question 3 (a) Solution:

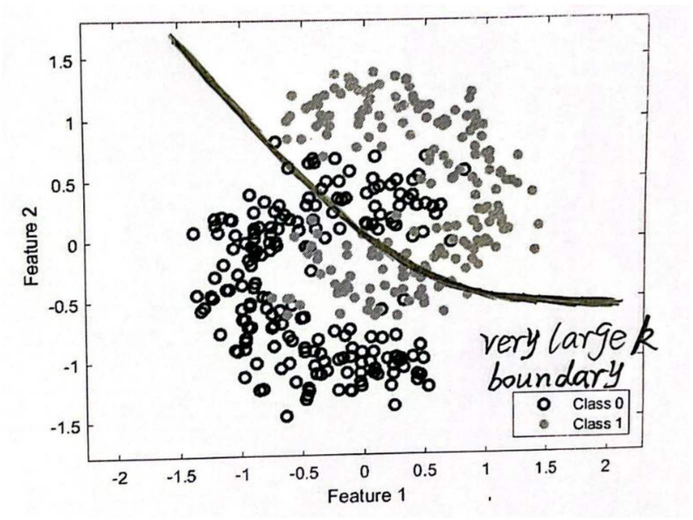
The sketch for an ideal boundary between the two classes is shown below.



Question 3 (b) Solution:

Two figures for different trends of k are shown below (left figure for very, very small k toward 1, right figure for very, very large k toward the number of observations, N).

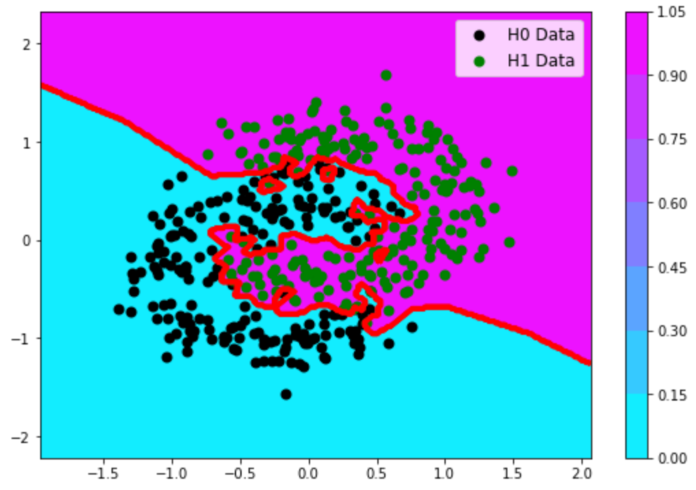




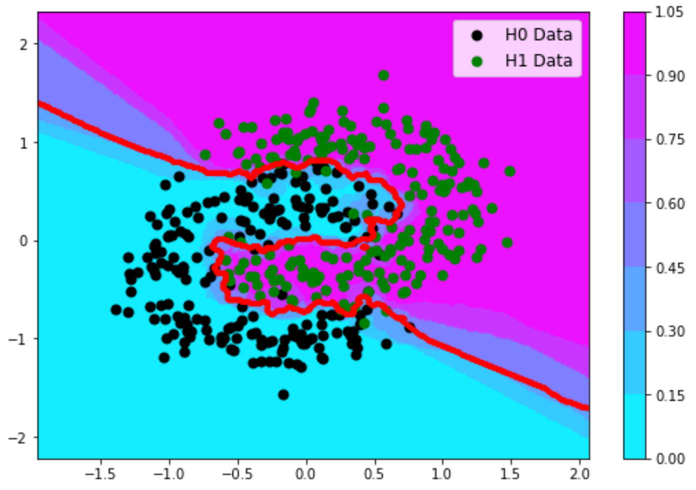
Basically, if k becomes very small toward 1, I expect the decision boundary to be more meandering compared with the ideal boundary, and the small k decision boundary could be so meandering that it tries to correctly classify each training point. If k becomes very large toward N, I expect the decision boundary to be gentler compared with the ideal boundary, and the large k decision boundary could be so gentle that it has very poor performance in classifying class 0 and class 1.

Question 4 (a) Solution:

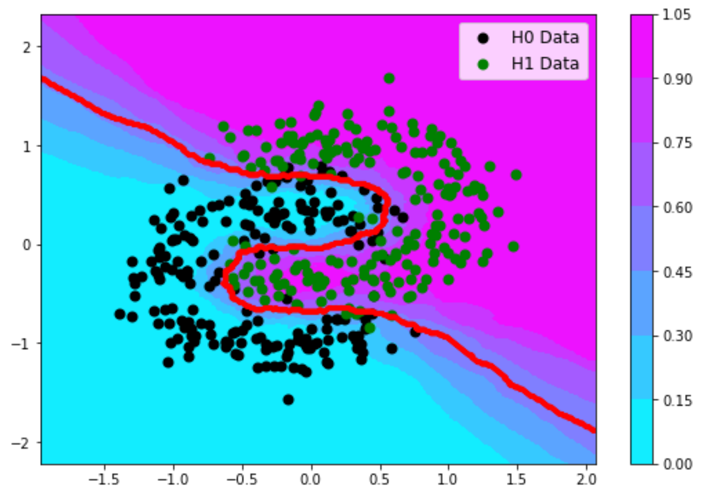
Note – The majority vote decision boundary is the red line.



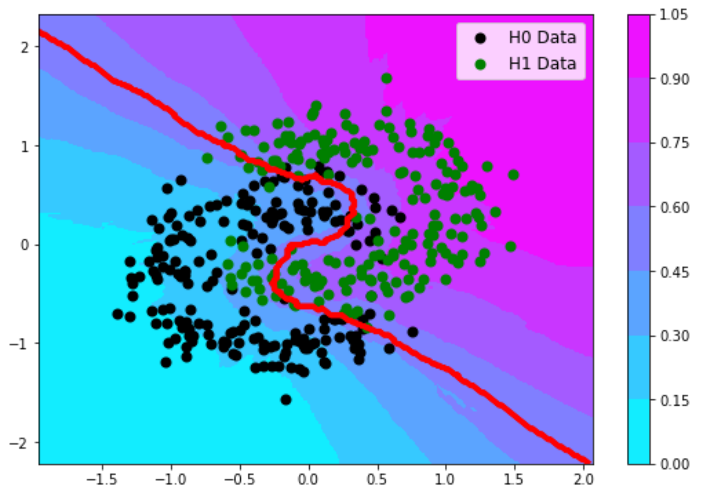
k = 1



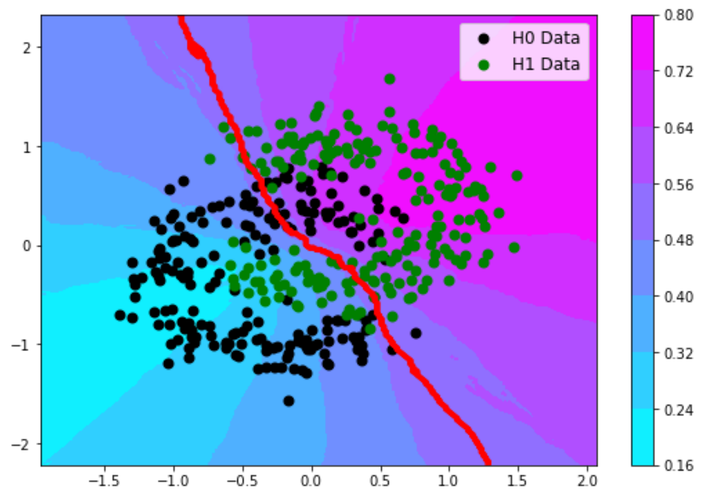
k = 5



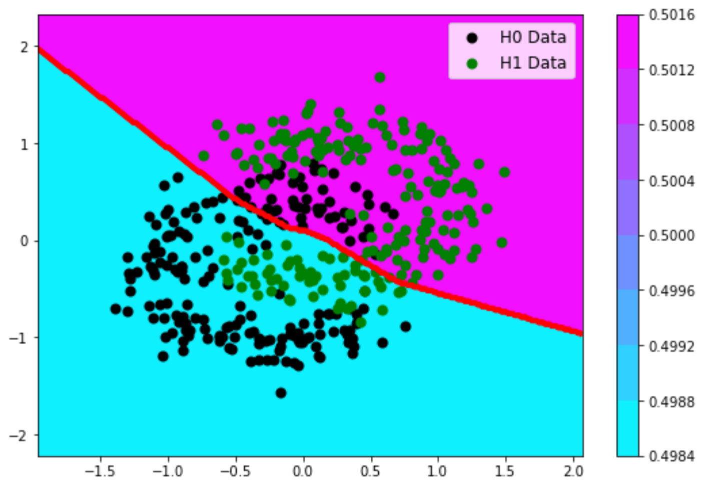
k = 31



k = 91



k = N / 2 – 1



k = N – 1

Question 4 (b) Solution:

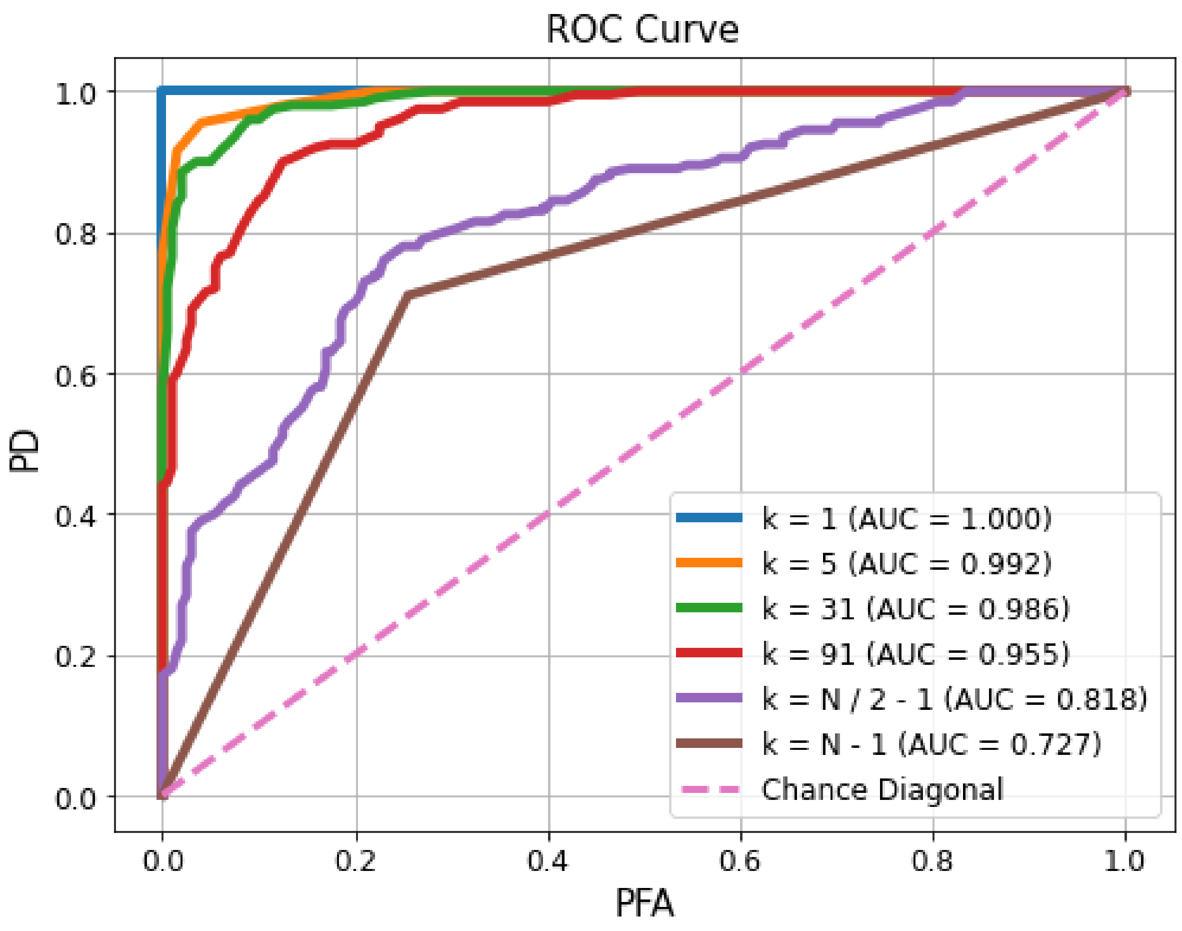
As k gets small ($k \rightarrow 1$), the KNN classifier considers a very small number of nearest neighbors for each classification (very “careful”), so the model has higher flexibility/complexity and lower simplicity, which is why the majority vote decision boundary is very meandering. Higher flexibility will lead to less systematic error (bias) for classification because the KNN classifier fits the training data so well, but high flexibility and low simplicity will also lead to high variance in model, based on the meandering majority vote decision boundary.

As k gets large ($k \rightarrow N$), the KNN classifier considers a very large number of nearest neighbors for each classification (very “careless”), so the model has lower flexibility/complexity and higher simplicity, which is why the majority vote decision boundary is very gentle (or flat). Lower flexibility will lead to more systematic error (bias) for classification because the KNN classifier is too simple, but low flexibility and high simplicity will also lead to low variance in model, based on the gentle majority vote decision boundary.

It should be noted that the question “how these deviations may be viewed as manifestations of bias or variance” has been answered in the above deviations explanations and analysis.

Question 5 (a):

When the KNN classifier is trained and tested on the training data, the plot for all ROCs is shown below.



When k = 1, max $P_{cd} = 1.0000$

When k = 5, max $P_{cd} = 0.9575$

When k = 31, max $P_{cd} = 0.9325$

When k = 91, max $P_{cd} = 0.8800$

When k = $\frac{N}{2} - 1$, max $P_{cd} = 0.7575$

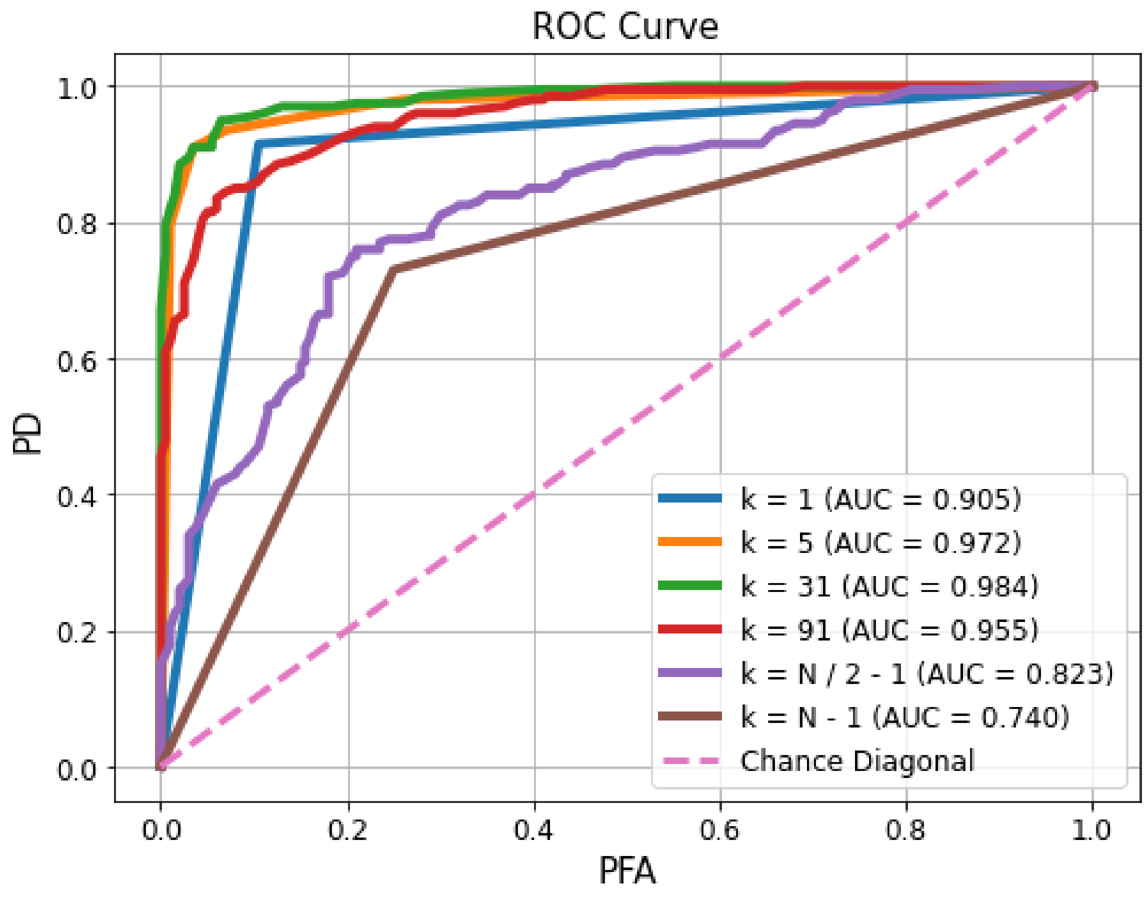
When k = $N - 1$, max $P_{cd} = 0.7275$

Question 5 (b) Solution:

When the KNN classifier is trained and tested on the training data, based only on these ROCs, and nothing else, I would recommend k = 1 to maximize P_{cd}, because when k = 1, the ROC curve sits exactly on the top-left corner, and has AUC = 1.0. These two facts mean that the KNN classifier achieves not only the best performance compared with other k values, but also the perfect classification performance, when k = 1.

Question 6 (a) Solution:

When the KNN classifier is trained on the training data and tested on separate testing data, the plot for all ROCs within the same axis is shown below.



When k = 1, max $P_{cd} = 0.9050$

When k = 5, max $P_{cd} = 0.9325$

When k = 31, max $P_{cd} = 0.9375$

When k = 91, max $P_{cd} = 0.8775$

When k = $\frac{N}{2} - 1$, max $P_{cd} = 0.7650$

When k = $N - 1$, max $P_{cd} = 0.7400$

Question 6 (b) Solution:

When the KNN classifier is trained on the training data and tested on separate testing data, based only on these ROCs, and nothing else, I would recommend k = 31 to maximize P_{cd}, because when k = 31, the ROC is closest to the top-left corner, and it has the largest AUC = 0.984. Therefore, I think k = 31 is the best choice to maximize P_{cd}, when the KNN classifier is trained on the training data and tested on separate testing data.

Question 7 (a):

To sample values of k between 1 and 399, I choose the following k values

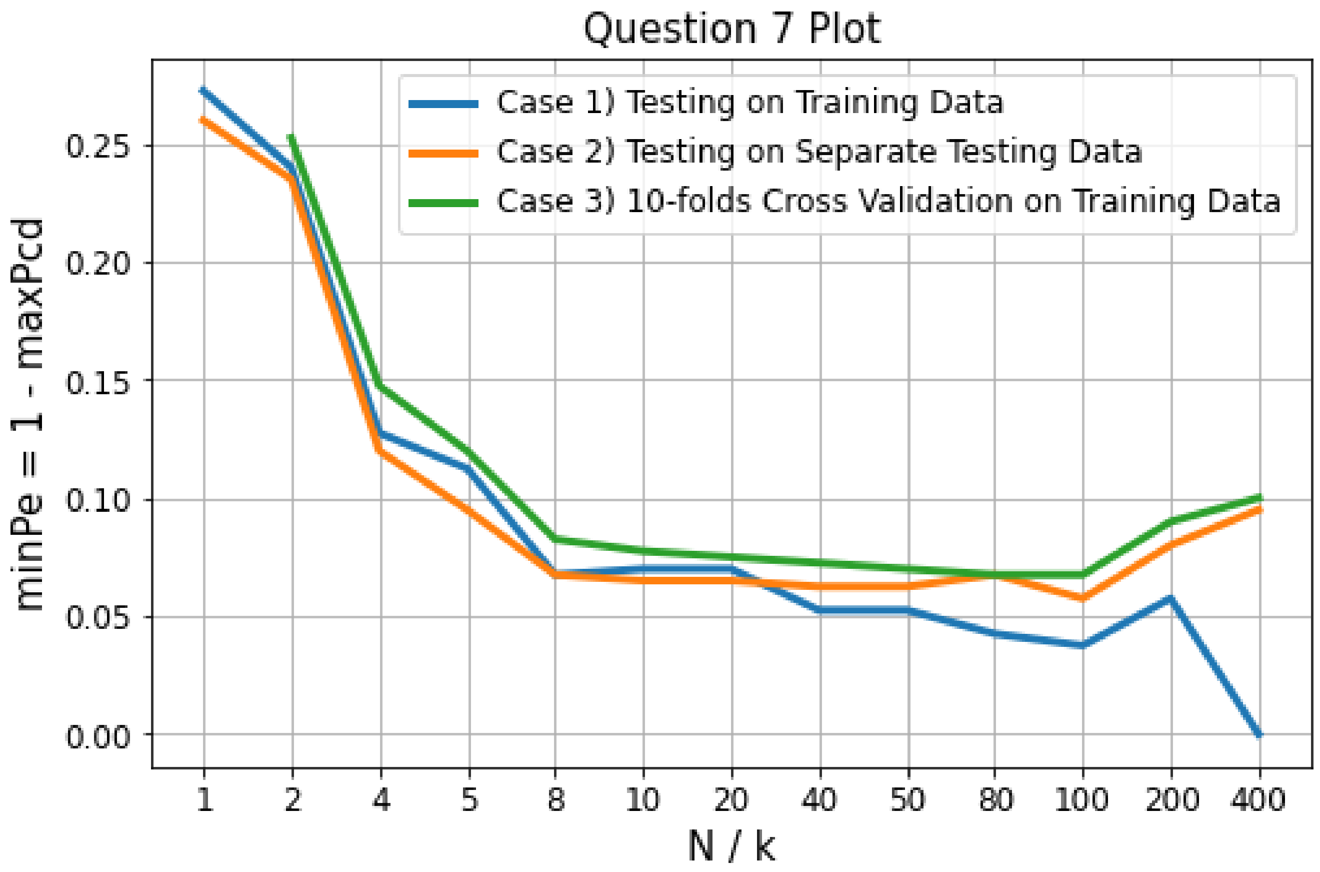
$$k = [1, 2, 4, 5, 8, 10, 20, 40, 50, 80, 100, 200, 399]$$

Correspondingly, the values for N/k (N = 400) should be

$$N / k = [400, 200, 100, 80, 50, 40, 20, 10, 8, 5, 4, 2, \frac{400}{399} \approx 1]$$

It should be noted that k = 399 will not be applied to Case 3 (10-folds cross-validation) because when we use 9 folds for training and 1-fold for testing each time, the maximum of training points is $(400/10) \times 9 = 360$, so that the k = 399 nearest neighbors can not be satisfied during training.

Question 7 (a) (i) – The plot for $minP_e$ $(= 1 - maxP_{cd})$ as a function of N/k is shown below.



Question 7 (a) (ii) – The 10-folds cross validation on training data has similar performance to testing on separate testing data. For a very large k (a very small N/k) value, the probability of error is very large, which means the systematic error (bias) is large, but a large k also means the model has low complexity/flexibility, which leads to small model variance as discussed in Question 4 (b). For k = 4 or k = 5 (small values but not the smallest), we can have the minimum probability of error, which means the systematic error (bias) is small, but the model variance (complexity/flexibility) also increases as we decrease k. For very small values k = 2 and k = 1 (very large $N/k = 200$ or $N/k = 400$), the probability of error increases instead, which means the KNN classifier encounters overfitting. At this point, even the model can have even less systematic error (bias) during training, the model complexity/flexibility (variance) is so high that overfitting makes the KNN classifier perform worse during testing (validation), compared with k = 4 or k = 5.

Question 7 (b) Solution:

Based on this graph, I would recommend = 4 ($N/k = 400/4 = 100$) to maximize P_{cd} (minimize P_e), because when k = 4, Case 1 (testing on training data) achieves the second smallest P_e, Case 2 (testing on separate testing data) and Case 3 (10-folds cross-validation on training data) both achieve the smallest P_e, these facts indicate that the KNN classifier with k = 4 can achieve the best performance max P_{cd} when predicting newly unseen data points. Assume I do not have separate test data available to me, I will refer to Case 3, and implement n-folds cross-validation on training data and evaluate min $P_e = 1 - \max P_{cd}$ for different k (N/k) values. Finally, I will find the k value which achieves the minimum probability of error and recommend this k value to maximize P_{cd}.

Question 8 Solution:

Reference the Packages/Toolboxes

I use numpy to process arraies and matrices.
I use pandas to read and process the original datasets.
I use matplotlib.pyplot to plot ROC curves and decision statistic surfaces.
To plot decision statistic surfaces, I also referenced the ECE580 course materials and the website below.
https://hackernoon.com/how-to-plot-a-decision-boundary-for-machine-learning-algorithms-in-python-3o1n3w07
I use metrics from sklearn to calculate area under curve (AUC).
I also use sklearn to implement KNN classifiers and cross-validation.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_validate
```

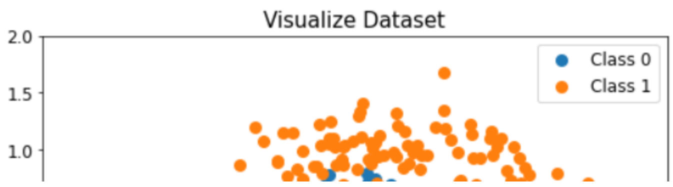
[2]:
```
filename1 = "dataSetHorseshoes.csv"
filename2 = "dataSetHorseshoesTest.csv"
dataframe1 = pd.read_csv(filename1, header = None)
dataframe2 = pd.read_csv(filename2, header = None)
```

[3]:
```
data = dataframe1.to_numpy()
testdata = dataframe2.to_numpy()
N = data.shape[0]
print(data.shape)
print(testdata.shape)
print(data[0:3, :])
```

```
(400, 3)
(400, 3)
[[ 1.     0.937  1.097]
 [ 0.    -1.144 -0.668]
 [ 1.     0.007 -0.344]]
```

[4]:
```
def visualize_dataset (data) :
    class0 = data[data[:,0] == 0., 1:]
    class1 = data[data[:,0] == 1., 1:]
    figure, axis = plt.subplots()
    axis.scatter(class0[:, 0], class0[:, 1], label = "Class 0", linewidths = 3)
    axis.scatter(class1[:, 0], class1[:, 1], label = "Class 1", linewidths = 3)
    axis.set_xlabel("Feature 1", fontsize = 15)
    axis.set_ylabel("Feature 2", fontsize = 15)
    axis.set_title("Visualize Dataset", fontsize = 15)
    #axis.grid()
    figure.set_size_inches(8, 6)
    plt.xticks(np.linspace(-2, 2, num = 9), fontsize = 12)
    plt.yticks(np.linspace(-2, 2, num = 9), fontsize = 12)
    plt.legend(fontsize = 12)
    plt.show()
    return None
```

```
[5]: visualize_dataset(data)
```

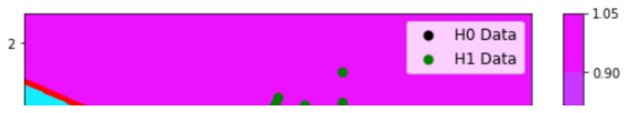


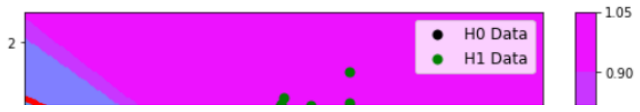
```
[6]: def majority_vote (k) :
         floor_int = int(np.floor(float(k) / 2))
         return (1 + floor_int)
     # print(majority_vote(1))
     # print(majority_vote(5))
     # print(majority_vote(31))
     # print(majority_vote(91))
```

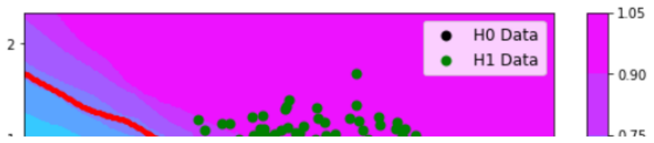
```
[7]: def knn_thres (k) :
         thres = np.zeros(k + 2)
         for i in range(k + 1) :
             thres[i] = float(i) / k
         thres[k + 1] = float("inf")
         return thres
     # print(knn_thres(5))
```

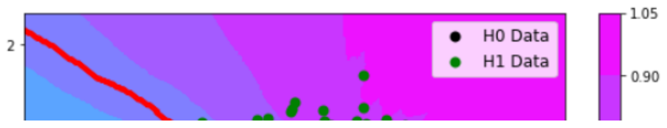
```
[8]: def find_mvdb (major_vote, zz) :
         mvdbx = []
         mvdby = []
         for i in range(zz.shape[0]) :
             for j in range(zz.shape[1]) :
                 if zz[i, j] == major_vote :
                     mvdbx.append(i)
                     mvdby.append(j)
         return np.array(mvdbx), np.array(mvdby)
```

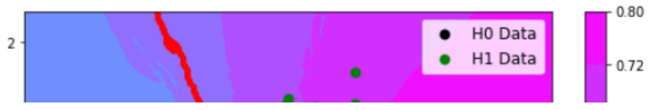
```
[9]: def visualize_dss (data, k) :
         x1max = np.max(data[:, 1])
         x1min = np.min(data[:, 1])
         x2max = np.max(data[:, 2])
         x2min = np.min(data[:, 2])
         x1Range = x1max - x1min
         x2Range = x2max - x2min
         x1 = np.linspace((x1min - (0.2 * x1Range)), (x1max + (0.2 * x1Range)), 251)
         x2 = np.linspace((x2min - (0.2 * x2Range)), (x2max + (0.2 * x2Range)), 251)
         xx, yy = np.meshgrid(x1, x2)
         r1, r2 = xx.flatten(), yy.flatten()
         r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
         grid = np.hstack((r1, r2))
         knnmodel = KNeighborsClassifier(n_neighbors = k)
         X = data[:, 1:]
         y = data[:, 0]
         knnmodel.fit(X, y)
         yhat = knnmodel.predict_proba(grid)
         yhat_H1 = yhat[:, 1]
         zz = yhat_H1.reshape(xx.shape)
         figure, axis = plt.subplots()
         c1 = axis.contourf(xx, yy, zz, cmap = mpl.cm.cool)
         plt.colorbar(c1)
         major_vote = float(majority_vote(k)) / k
         c2 = axis.contour(xx, yy, zz >= major_vote, colors = "r", linewidths = 3)
         # mvdbx, mvdby = find_mvdb(major_vote, zz)
         # axis.plot(xx[0, mvdbx], yy[mvdby, 0], label = "Majority Vote Decision Boundary", linewidth = 1, c = "r")
         axis.scatter(data[data[:,0]==0., 1], data[data[:,0]==0., 2], label = "H0 Data", linewidths = 2, c = "k")
         axis.scatter(data[data[:,0]==1., 1], data[data[:,0]==1., 2], label = "H1 Data", linewidths = 2, c = "g")
         figure.set_size_inches((9, 6))
         plt.xticks(fontsize = 10)
         plt.yticks(fontsize = 10)
         plt.legend(fontsize = 12)
         plt.show()
         return None
```

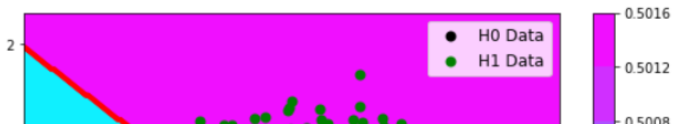
[10]:
```
visualize_dss(data, k = 1)
```


[11]:
```
visualize_dss(data, k = 5)
```


[12]:
```
visualize_dss(data, k = 31)
```


[13]:
```
visualize_dss(data, k = 91)
```


[14]:
```
visualize_dss(data, k = (int(N / 2) - 1))
```


[15]:
```
visualize_dss(data, k = (N - 1))
```


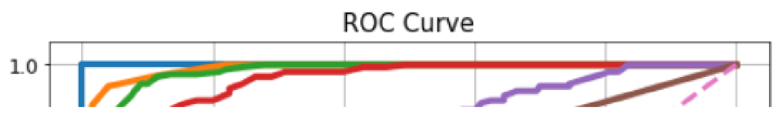
[16]:
```
def divide_labels (data) :
    npdata = np.copy(data)
    data0 = npdata[npdata[:,0] == 0., :]
    data1 = npdata[npdata[:,0] == 1., :]
    return data0, data1

def compute_PFA (H0, thres) :
    H0_ds = np.sort(H0[:, 1])
    false_alarm = float(len(H0_ds[H0_ds >= thres])) / (H0.shape[0])
    return false_alarm

def compute_PD (H1, thres) :
    H1_ds = np.sort(H1[:, 1])
    detection = float(len(H1_ds[H1_ds >= thres])) / (H1.shape[0])
    return detection
```

[17]:
```
def compute_ROC (arr_data, knnthres) :
    arr_PD = np.zeros(len(knnthres))
    arr_PFA = np.zeros(len(knnthres))
    H0, H1 = divide_labels(arr_data)
    for i in range(len(knnthres)) :
        arr_PD[i] = compute_PD(H1, knnthres[i])
        arr_PFA[i] = compute_PFA(H0, knnthres[i])
    return np.flipud(arr_PD), np.flipud(arr_PFA)
```

[18]:
```
def plot_ROC (arr_data, arr_k) :
    s1_pd, s1_pfa = compute_ROC(arr_data[0], knn_thres(arr_k[0]))
    s2_pd, s2_pfa = compute_ROC(arr_data[1], knn_thres(arr_k[1]))
    s3_pd, s3_pfa = compute_ROC(arr_data[2], knn_thres(arr_k[2]))
    s4_pd, s4_pfa = compute_ROC(arr_data[3], knn_thres(arr_k[3]))
    s5_pd, s5_pfa = compute_ROC(arr_data[4], knn_thres(arr_k[4]))
    s6_pd, s6_pfa = compute_ROC(arr_data[5], knn_thres(arr_k[5]))
    auc1 = metrics.auc(s1_pfa, s1_pd)
    auc2 = metrics.auc(s2_pfa, s2_pd)
    auc3 = metrics.auc(s3_pfa, s3_pd)
    auc4 = metrics.auc(s4_pfa, s4_pd)
    auc5 = metrics.auc(s5_pfa, s5_pd)
    auc6 = metrics.auc(s6_pfa, s6_pd)
    refln = np.linspace(0, 1, num = 101)
    figure, axis = plt.subplots()
    axis.plot(s1_pfa, s1_pd, label = "k = 1 (AUC = %.3f)" % auc1, linewidth = 4)
    axis.plot(s2_pfa, s2_pd, label = "k = 5 (AUC = %.3f)" % auc2, linewidth = 4)
    axis.plot(s3_pfa, s3_pd, label = "k = 31 (AUC = %.3f)" % auc3, linewidth = 4)
    axis.plot(s4_pfa, s4_pd, label = "k = 91 (AUC = %.3f)" % auc4, linewidth = 4)
    axis.plot(s5_pfa, s5_pd, label = "k = N / 2 - 1 (AUC = %.3f)" % auc5, linewidth = 4)
    axis.plot(s6_pfa, s6_pd, label = "k = N - 1 (AUC = %.3f)" % auc6, linewidth = 4)
    axis.plot(refln, refln, "--", label = "Chance Diagonal", linewidth = 3)
    axis.set_xlabel("PFA", fontsize = 15)
    axis.set_ylabel("PD", fontsize = 15)
    axis.set_title("ROC Curve", fontsize = 15)
    axis.grid()
    figure.set_size_inches(8, 6)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.legend(fontsize = 12)
    plt.show()
    return None
```

[19]:
```
arr_k = np.array([1, 5, 31, 91, int(N/2)-1, N-1])
arr_data5 = np.zeros((len(arr_k), data.shape[0], 2))
for i in range(len(arr_k)) :
    arr_data5[i, :, 0] = data[:, 0]

model51 = KNeighborsClassifier(n_neighbors = arr_k[0])
model52 = KNeighborsClassifier(n_neighbors = arr_k[1])
model53 = KNeighborsClassifier(n_neighbors = arr_k[2])
model54 = KNeighborsClassifier(n_neighbors = arr_k[3])
model55 = KNeighborsClassifier(n_neighbors = arr_k[4])
model56 = KNeighborsClassifier(n_neighbors = arr_k[5])

model51.fit(data[:, 1:], data[:, 0])
model52.fit(data[:, 1:], data[:, 0])
model53.fit(data[:, 1:], data[:, 0])
model54.fit(data[:, 1:], data[:, 0])
model55.fit(data[:, 1:], data[:, 0])
model56.fit(data[:, 1:], data[:, 0])

A = model51.predict_proba(data[:, 1:])
B = model52.predict_proba(data[:, 1:])
C = model53.predict_proba(data[:, 1:])
D = model54.predict_proba(data[:, 1:])
E = model55.predict_proba(data[:, 1:])
F = model56.predict_proba(data[:, 1:])

arr_data5[0, :, 1] = A[:, 1]
arr_data5[1, :, 1] = B[:, 1]
arr_data5[2, :, 1] = C[:, 1]
arr_data5[3, :, 1] = D[:, 1]
arr_data5[4, :, 1] = E[:, 1]
arr_data5[5, :, 1] = F[:, 1]
```

[20]:
```
plot_ROC(arr_data5, arr_k)
```

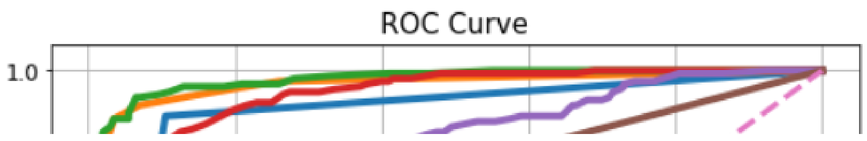


[21]:
```
print("%.4f" % model51.score(data[:, 1:], data[:, 0]))
print("%.4f" % model52.score(data[:, 1:], data[:, 0]))
print("%.4f" % model53.score(data[:, 1:], data[:, 0]))
print("%.4f" % model54.score(data[:, 1:], data[:, 0]))
print("%.4f" % model55.score(data[:, 1:], data[:, 0]))
print("%.4f" % model56.score(data[:, 1:], data[:, 0]))
```

```
1.0000
0.9575
0.9325
0.8800
0.7575
0.7275
```

[22]:
```
arr_k = np.array([1, 5, 31, 91, int(N/2)-1, N-1])
arr_data6 = np.zeros((len(arr_k), testdata.shape[0], 2))
for i in range(len(arr_k)) :
    arr_data6[i, :, 0] = testdata[:, 0]

model61 = KNeighborsClassifier(n_neighbors = arr_k[0])
model62 = KNeighborsClassifier(n_neighbors = arr_k[1])
model63 = KNeighborsClassifier(n_neighbors = arr_k[2])
model64 = KNeighborsClassifier(n_neighbors = arr_k[3])
model65 = KNeighborsClassifier(n_neighbors = arr_k[4])
model66 = KNeighborsClassifier(n_neighbors = arr_k[5])

model61.fit(data[:, 1:], data[:, 0])
model62.fit(data[:, 1:], data[:, 0])
model63.fit(data[:, 1:], data[:, 0])
model64.fit(data[:, 1:], data[:, 0])
model65.fit(data[:, 1:], data[:, 0])
model66.fit(data[:, 1:], data[:, 0])

A = model61.predict_proba(testdata[:, 1:])
B = model62.predict_proba(testdata[:, 1:])
C = model63.predict_proba(testdata[:, 1:])
D = model64.predict_proba(testdata[:, 1:])
E = model65.predict_proba(testdata[:, 1:])
F = model66.predict_proba(testdata[:, 1:])

arr_data6[0, :, 1] = A[:, 1]
arr_data6[1, :, 1] = B[:, 1]
arr_data6[2, :, 1] = C[:, 1]
arr_data6[3, :, 1] = D[:, 1]
arr_data6[4, :, 1] = E[:, 1]
arr_data6[5, :, 1] = F[:, 1]
```

[23]:
```
plot_ROC(arr_data6, arr_k)
```



[24]:
```
print("%.4f" % model61.score(testdata[:, 1:], testdata[:, 0]))
print("%.4f" % model62.score(testdata[:, 1:], testdata[:, 0]))
print("%.4f" % model63.score(testdata[:, 1:], testdata[:, 0]))
print("%.4f" % model64.score(testdata[:, 1:], testdata[:, 0]))
print("%.4f" % model65.score(testdata[:, 1:], testdata[:, 0]))
print("%.4f" % model66.score(testdata[:, 1:], testdata[:, 0]))
```

```
0.9050
0.9325
0.9375
0.8775
0.7650
0.7400
```

[25]:
```
###########################################################################
```

[26]:
```
print(data.shape)
print(testdata.shape)
print(data[0:3, :])
```

```
(400, 3)
(400, 3)
[[ 1.     0.937  1.097]
 [ 0.    -1.144 -0.668]
 [ 1.     0.007 -0.344]]
```

[27]:
```
sample_k = np.array([1, 2, 4, 5, 8, 10, 20, 40, 50, 80, 100, 200, 399]).astype(int)
print(sample_k.shape)
print(sample_k)
N_divide_k = (N / sample_k).astype(int)
print(N_divide_k.shape)
print(N_divide_k)
```

```
(13,)
[  1   2   4   5   8  10  20  40  50  80 100 200 399]
(13,)
[400 200 100  80  50  40  20  10   8   5   4   2   1]
```

[ ]:
```
# Block 28 is only for incremental testing
```

[28]:
```
cvmodel = KNeighborsClassifier(n_neighbors = 5)
cvresults = cross_validate(estimator = cvmodel,
                           X = data[:, 1:], y = data[:, 0], cv = 10)
print(type(cvresults))
print(cvresults.keys())
print(cvresults["test_score"])
print(type(cvresults["test_score"]))
print("%.4f" % np.mean(cvresults["test_score"]))
```

```
<class 'dict'>
dict_keys(['fit_time', 'score_time', 'test_score'])
[0.925 0.9   0.975 0.925 0.925 0.925 0.9   0.975 0.925 0.95 ]
<class 'numpy.ndarray'>
0.9325
```

[29]:
```
def compute_Pe (sample_k, case) :
    Pe = np.zeros(sample_k.shape[0])
    for i in range(sample_k.shape[0]) :
        model = KNeighborsClassifier(n_neighbors = sample_k[i])
        if case == 1 :
            model.fit(data[:, 1:], data[:, 0])
            Pcd = model.score(data[:, 1:], data[:, 0])
            Pe[i] = 1 - Pcd
        if case == 2 :
            model.fit(data[:, 1:], data[:, 0])
            Pcd = model.score(testdata[:, 1:], testdata[:, 0])
            Pe[i] = 1 - Pcd
        if case == 3 :
            if sample_k[i] != np.max(sample_k) :
                cvresults = cross_validate(estimator = model, X = data[:, 1:], y = data[:, 0], cv = 10)
                Pcd = np.mean(cvresults["test_score"])
                Pe[i] = 1 - Pcd
            else :
                Pe[i] = np.nan
    return Pe
```

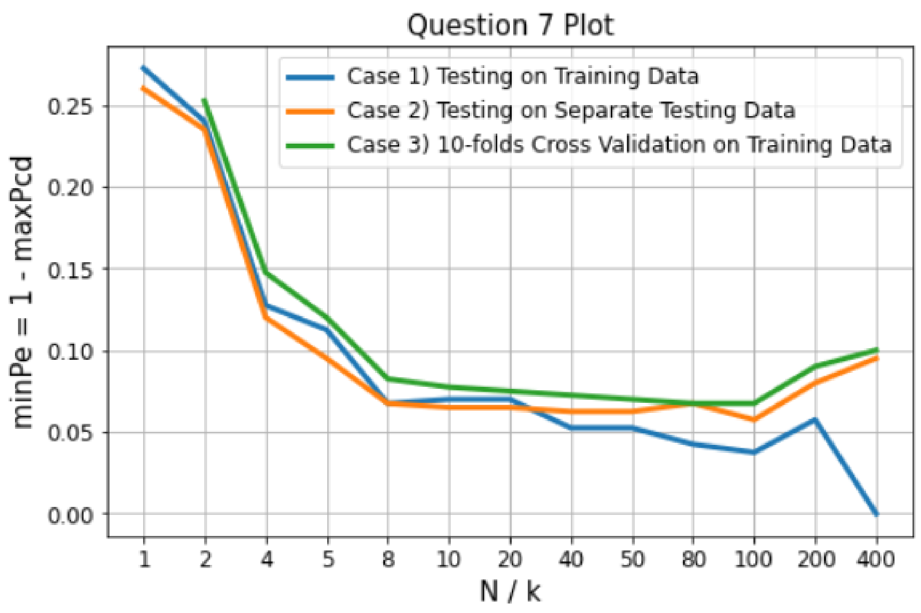
```
[30]: def plot_Pe (Pe1, Pe2, Pe3, N_k) :
          figure, axis = plt.subplots()
          xaxis = np.flipud(N_k)
          adjustx = np.arange(xaxis.shape[0])
          axis.plot(adjustx, np.flipud(Pe1), label = "Case 1) Testing on Training Data", linewidth = 3)
          axis.plot(adjustx, np.flipud(Pe2), label = "Case 2) Testing on Separate Testing Data", linewidth = 3)
          axis.plot(adjustx, np.flipud(Pe3), label = "Case 3) 10-folds Cross Validation on Training Data", linewidth = 3)
          axis.set_xlabel("N / k", fontsize = 15)
          axis.set_ylabel("minPe = 1 - maxPcd", fontsize = 15)
          axis.set_title("Question 7 Plot", fontsize = 15)
          axis.grid()
          figure.set_size_inches(8, 5)
          plt.xticks(adjustx, xaxis, fontsize = 12)
          plt.yticks(fontsize = 12)
          plt.legend(fontsize = 12)
          plt.show()
          return None
```

```
[31]: Pe_case1 = compute_Pe(sample_k, case = 1)
```

```
[32]: Pe_case2 = compute_Pe(sample_k, case = 2)
```

```
[33]: Pe_case3 = compute_Pe(sample_k, case = 3)
```

```
[34]: plot_Pe(Pe_case1, Pe_case2, Pe_case3, N_divide_k)
```


Question 7 Plot
Case 1) Testing on Training Data
Case 2) Testing on Separate Testing Data
Case 3) 10-folds Cross Validation on Training Data
minPe = 1 - maxPcd
N / k