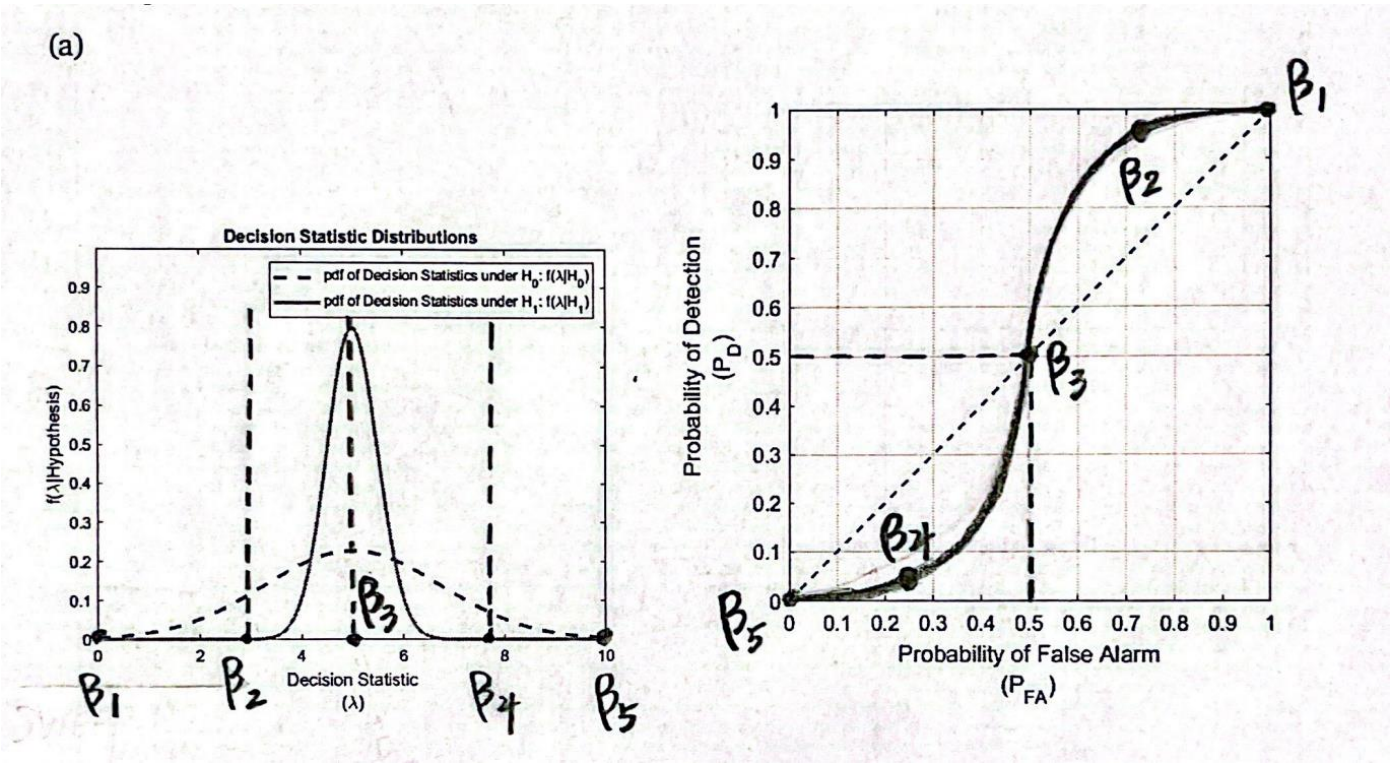
ECE 580 Homework #3

Libo Zhang (lz200)

Decision Statistics – ROCs Section

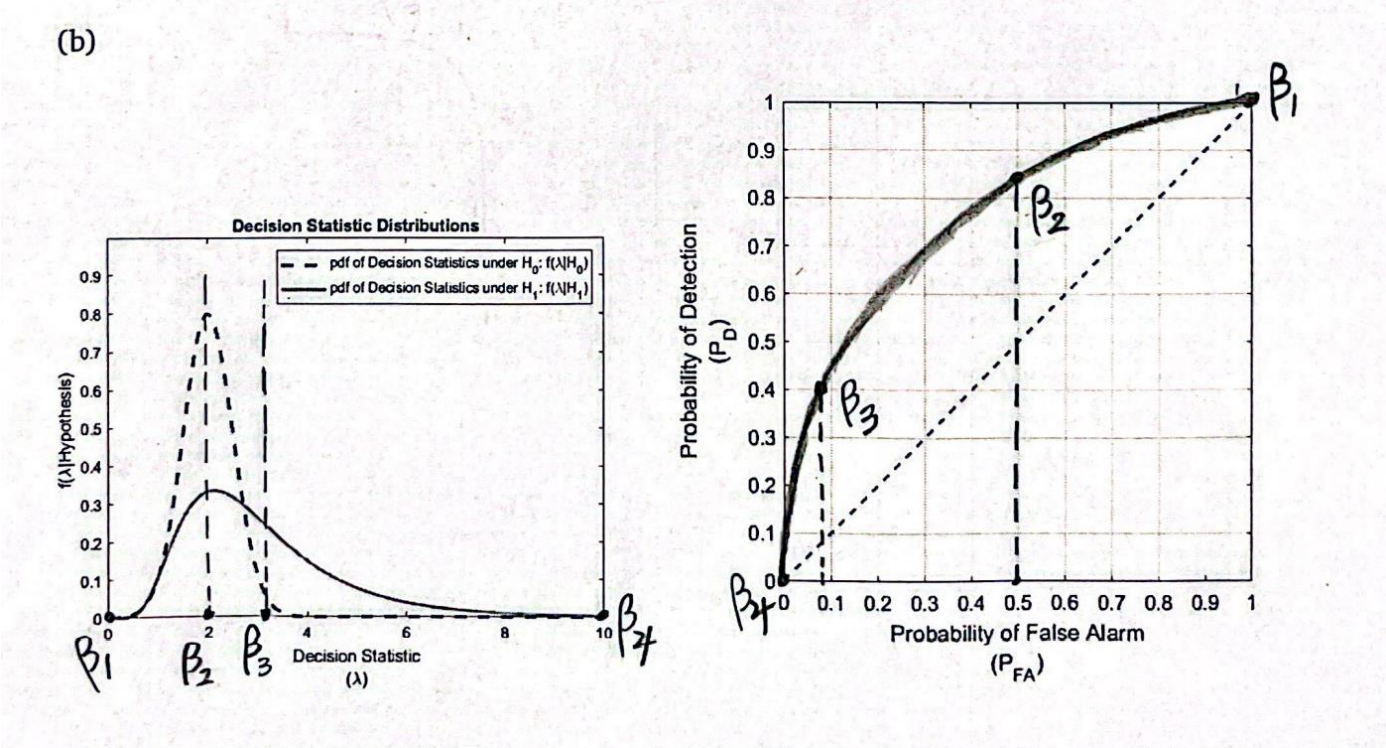
Question 1 (a) Solution:



Explain how you determined the shape and slope of the ROC as a function of λ, as well as how you determined key points on the ROC, such as PD = 0.5 or PFA = 0.5.

Explanation 1 (a) – when sweeping threshold across the decision statistic (λ) from β_1 to β_5, first we lose more PFA, then in the middle part we lose more PD, and finally we lose more PFA. We lose too much PD in the middle sweeping stage, so the ROC drops below the chance diagonal after PD = PFA = 0.5. The reason for determining the key operating point (PFA = 0.5, PD = 0.5) is that the decision statistic distributions for both H0 and H1 are symmetric, so PD and PFA will be 0.5 at the same threshold, which is β_3.

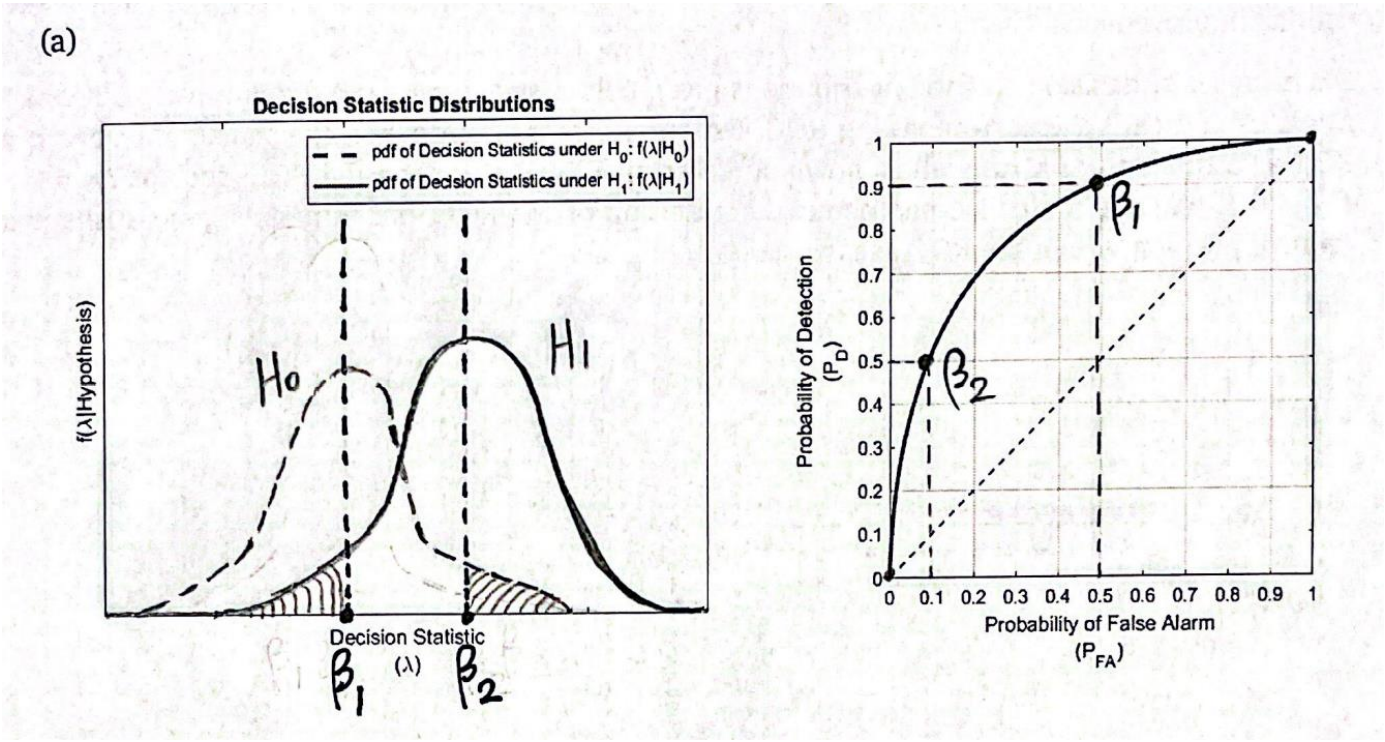
Question 1 (b) Solution:



Explain how you determined the shape and slope of the ROC as a function of λ, as well as how you determined key points on the ROC, such as PD = 0.5 or PFA = 0.5.

Explanation 1 (b) – When sweeping threshold across the decision statistic (λ) from β_1 to β_4, firstly we lose more PFA, and then we lose more PD. We lose too much PFA in the beginning so the ROC can lie entirely above the chance diagonal. To determine key points, when PFA = 0.5 (β_2), we can see that PD > 0.5. When PD = 0.5, the sweeping threshold should be located between β_2 and β_3, and we can see that PFA is approaching 0.

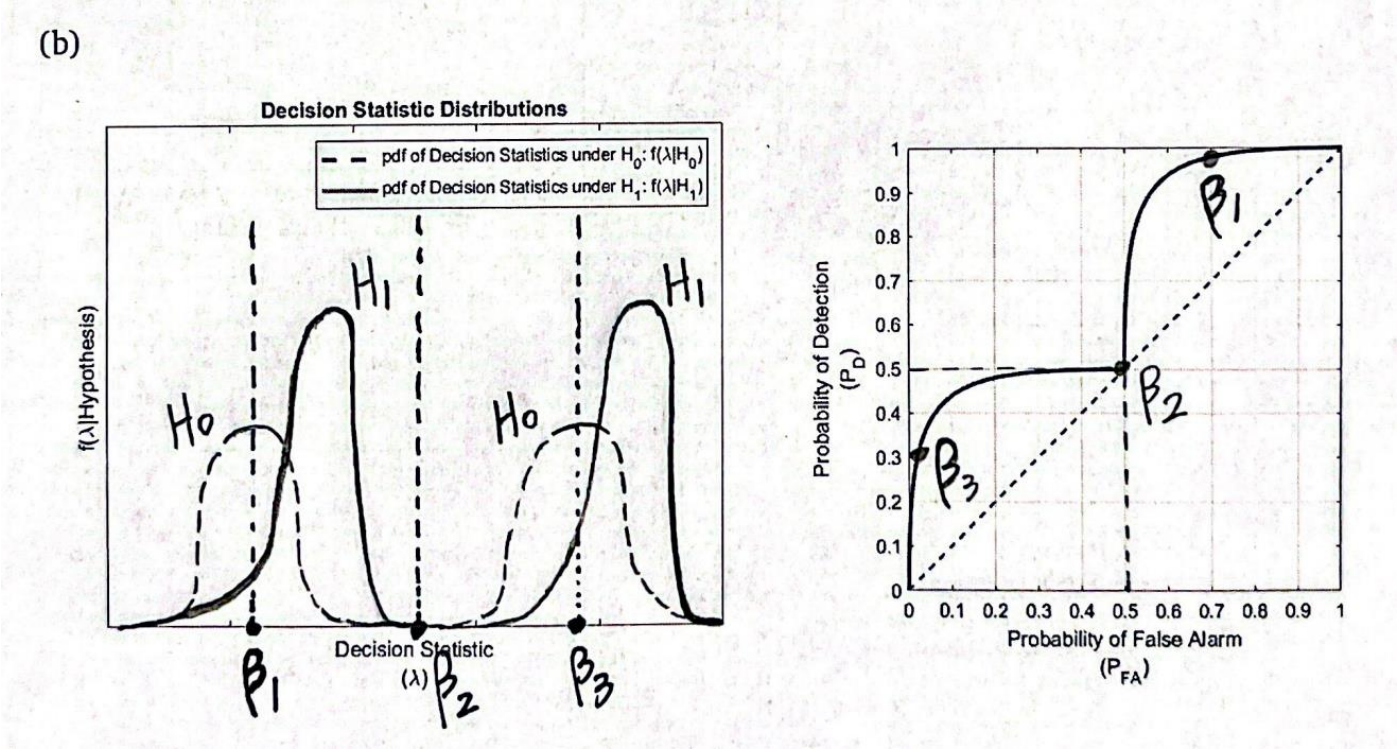
Question 2 (a) Solution:



Explain how you determined the shapes of the decision statistic distributions as a function of λ.

Explanation 2 (a) – Based on the ROC, when sweeping threshold from less than the minimum of decision statistic (PFA = PD = 1) to greater than the maximum of decision statistic (PFA = PD = 0), firstly, we lose more PFA. Secondly, we lose more PD. We can also see that the ROC lies entirely above the chance diagonal. These factors are actually very common when we are doing binary classification and building ROC, and the PDFs for H0 and H1 could simply be two uniform distributions with some intersection. Therefore, I think the shapes of decision statistic distributions should look like the above sketch.

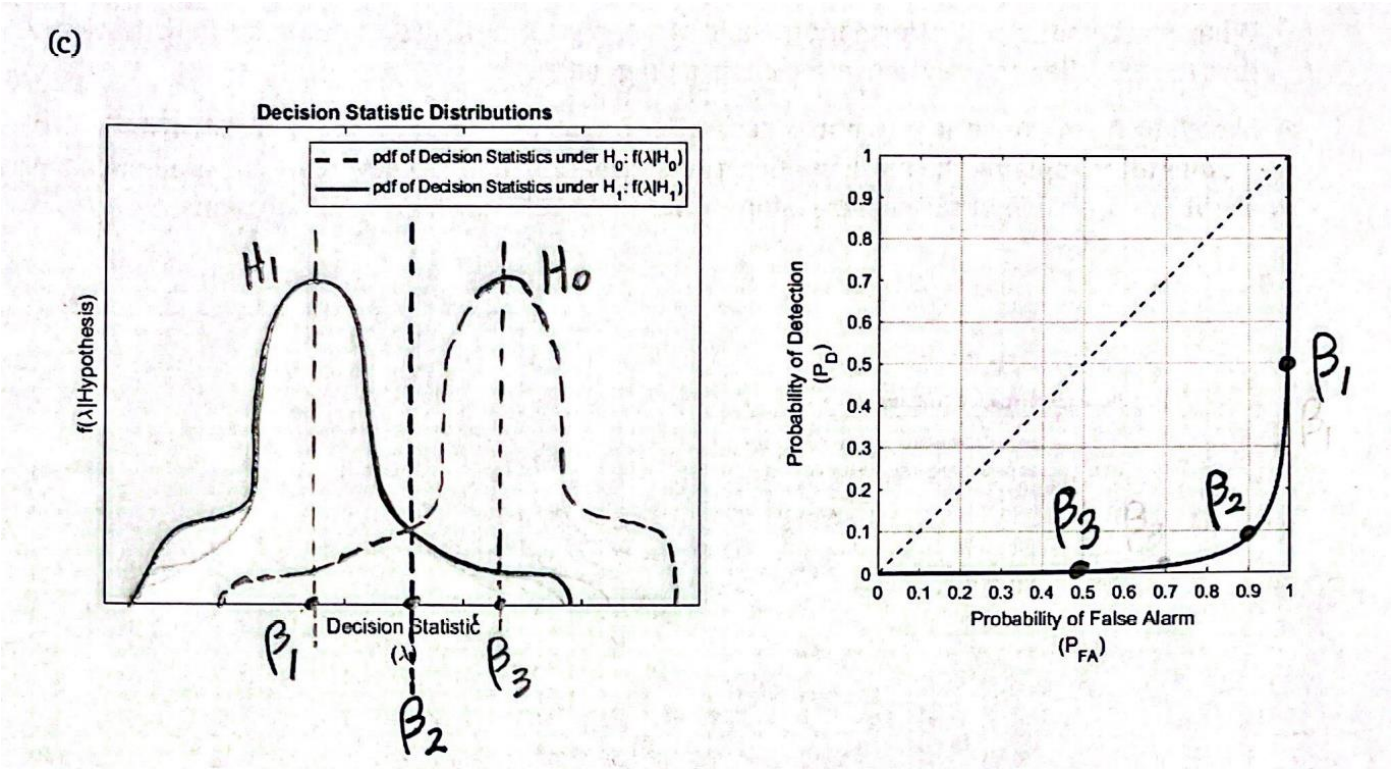
Question 2 (b) Solution:



Explain how you determined the shapes of the decision statistic distributions as a function of λ.

Explanation 2 (b) – Based on the ROC, when sweeping threshold from less than the minimum of decision statistic (PFA = PD = 1) to greater than the maximum of decision statistic (PFA = PD = 0), firstly, we lose more PFA, and then we lose more PD. After passing (PFA = 0.5, PD = 0.5, β_2) operating point, again we lose more PFA first and then lose more PD. We can also see that the ROC lies entirely above the chance diagonal. Based on the above trends, first I think the decision statistic distributions should be symmetric for both H0 and H1, and the distributions look like being divided into two parts by β_2, and the divided two parts should have the same shape. Therefore, I think the shapes should look like the above sketch.

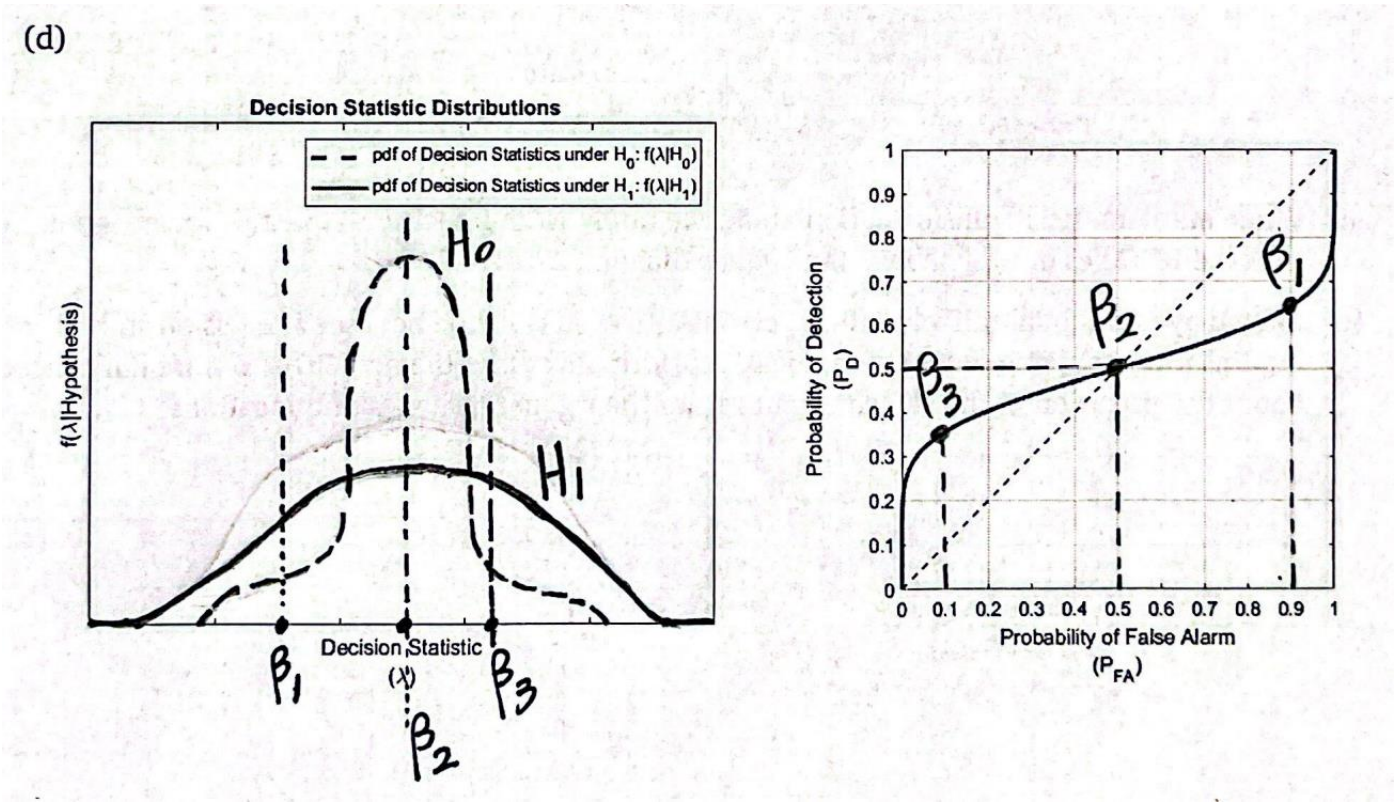
Question 2 (c) Solution:



Explain how you determined the shapes of the decision statistic distributions as a function of λ.

Explanation 2 (c) – The given ROC lies entirely below the chance diagonal. This means that when sweeping the threshold from less than the minimum of decision statistic (PFA = PD = 1) to greater than the maximum of decision statistic (PFA = PD = 0), we lose too much PD in the beginning, and after losing almost all PD, we start to lose more PFA. This ROC is also very typical, so that I think the shapes of the decision statistic distributions for H0 and H1 can be represented by the above sketch. It should be noted that I do want to sketch two uniform distributions for both H0 and H1, but I found it a bit difficult.

Question 2 (d) Solution:



Explain how you determined the shapes of the decision statistic distributions as a function of λ.

Explanation 2 (d) – Based on the given ROC, when sweeping threshold from less than the minimum of decision statistic (PFA = PD = 1) to greater than the maximum of decision statistic (PFA = PD = 0), first we lose more PD, then in the middle sweeping stage we lose more PFA, and finally we end by losing more PD. Since the ROC passes the operating point (PFA = PD = 0.5, β_2), we know that the decision statistic distributions for both H0 and H1 should be symmetric along threshold β_2. And based on the PD/PFA/PD losing order, I think the shapes of the decision statistic distributions for H0 and H1 should look like the above sketch.

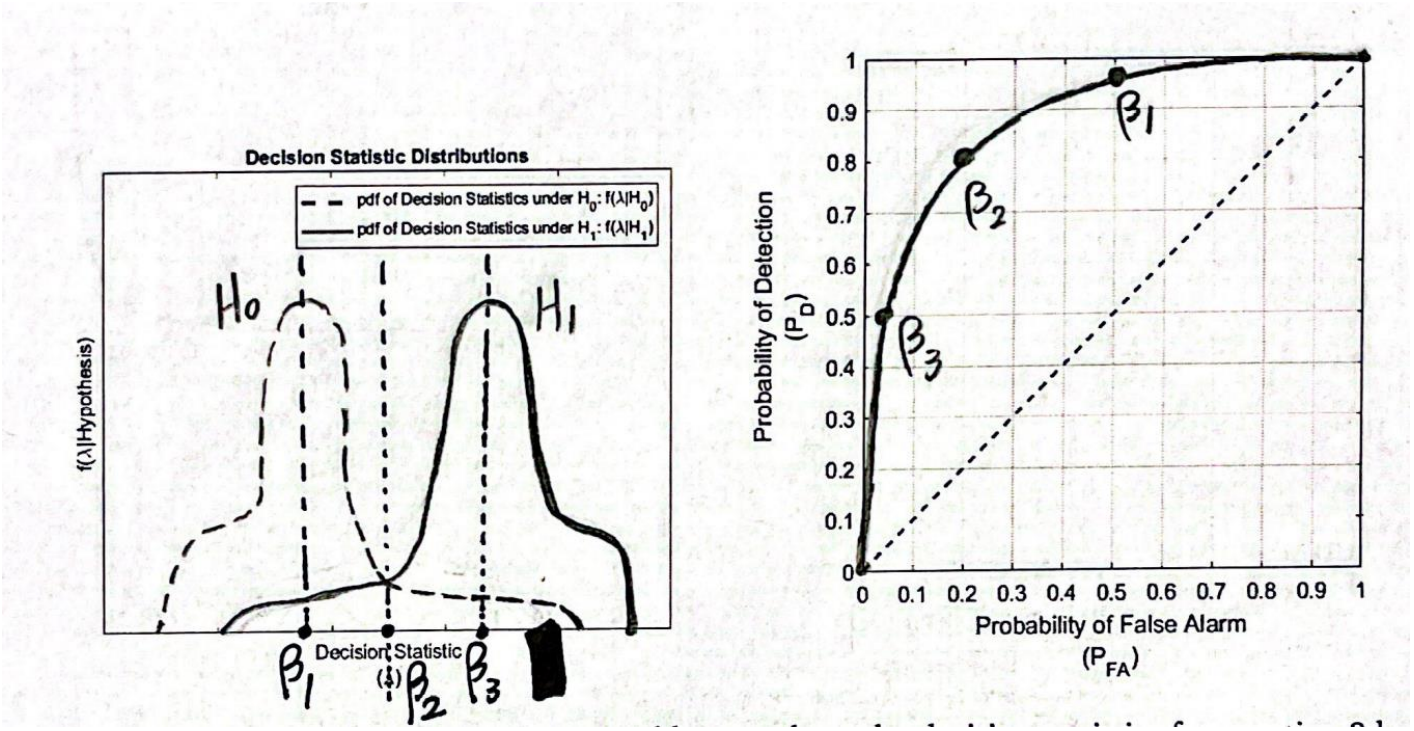
Question 3 (a) Solution:

In Question 2 (c), the ROC lies entirely below the chance diagonal. To make the ROC lie entirely above the chance diagonal, we should directly reverse decision rule, or directly transform all decision statistics.

The formal mathematical transformation is defined below.

$$\lambda_{new} = (-1)\lambda_{old}, \forall \lambda_{old}$$

Question 3 (b) Solution:



Question 3 (c) Solution:

In Question 2 (d), only the right half part of ROC ($0.5 < P_{FA} \leq 1$) lies below the chance diagonal, while the left half part of ROC ($0 \leq P_{FA} \leq 0.5$) already lies above the chance diagonal. Therefore, to make the ROC entirely lie above the chance diagonal, we should first set some bounding constraints to exclude the left half part of ROC, then we reverse decision rule or transform decision statistics for the right half part of ROC, with the purpose of slowing down the drop of P_D (preventing ROC from dropping below the chance diagonal).

Since the ROC passes the point (PFA = 0.5, PD = 0.5), we know that the original decision statistic distributions for H0 and H1 are symmetric. We first find the threshold β_2 which leads to PFA = 0.5 and PD = 0.5.

For all decision statistics (both H0 and H1 considered) greater than β_2, we keep them unchanged (leave the left half part of ROC unchanged).

For all decision statistics (both H0 and H1 considered) smaller than β_2, we reverse decision rule or transform these decision statistics (flip the right half part of ROC so that it lies entirely above the chance diagonal).

The formal mathematical transformation is defined below.

$$\lambda_{old} = [\lambda_{old-SmallerPart}, \lambda_{old-LargerPart}]$$

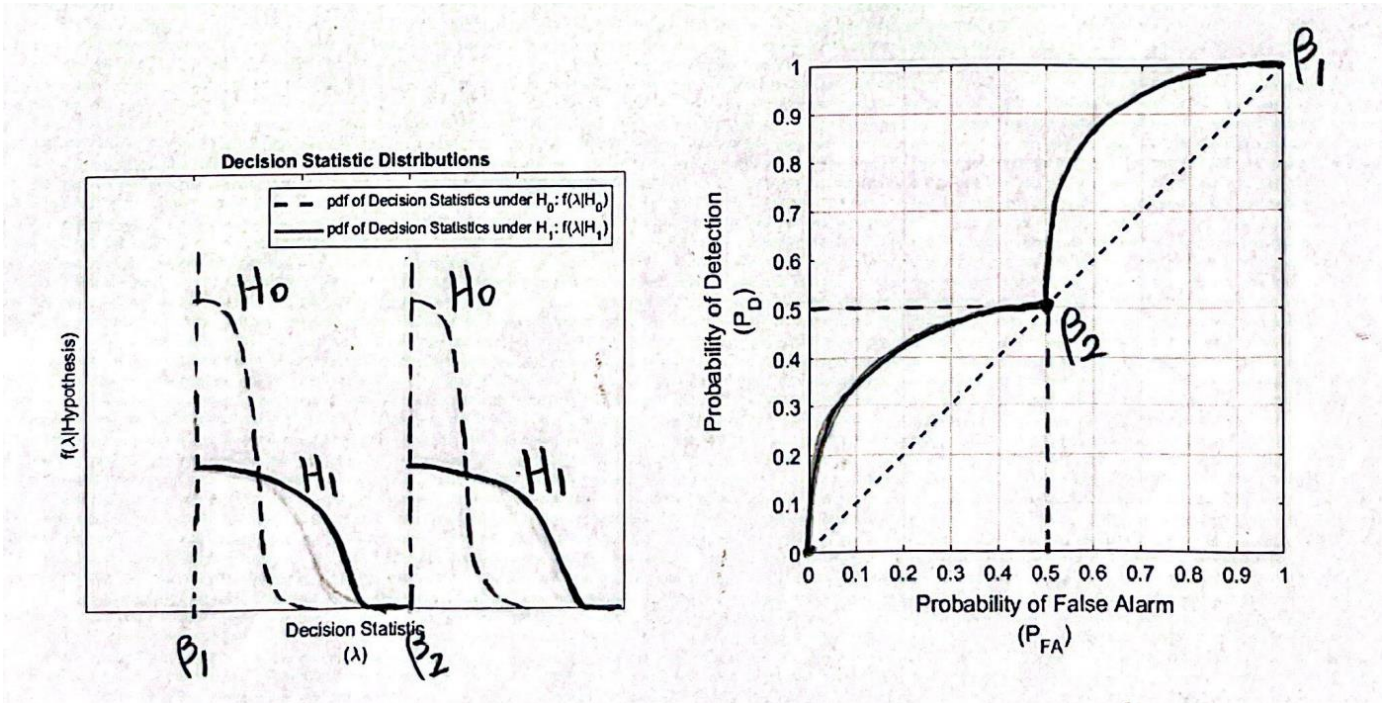
$$\exists \beta_2, such\ that\ \{\beta_2 > \lambda_{old-SmallerPart}\ and\ \beta_2 \leq \lambda_{old-LargerPart}\ and\ (P_{FA} = 0.5, P_D = 0.5, \beta_2)\}$$

$$\lambda_{new-SmallerPart} = (-1)\lambda_{old-SmallerPart}\quad (Reverse\ Decision\ Rule)$$

$$\lambda_{new-LargerPart} = \lambda_{old-LargerPart}\quad (Keep\ Unchanged)$$

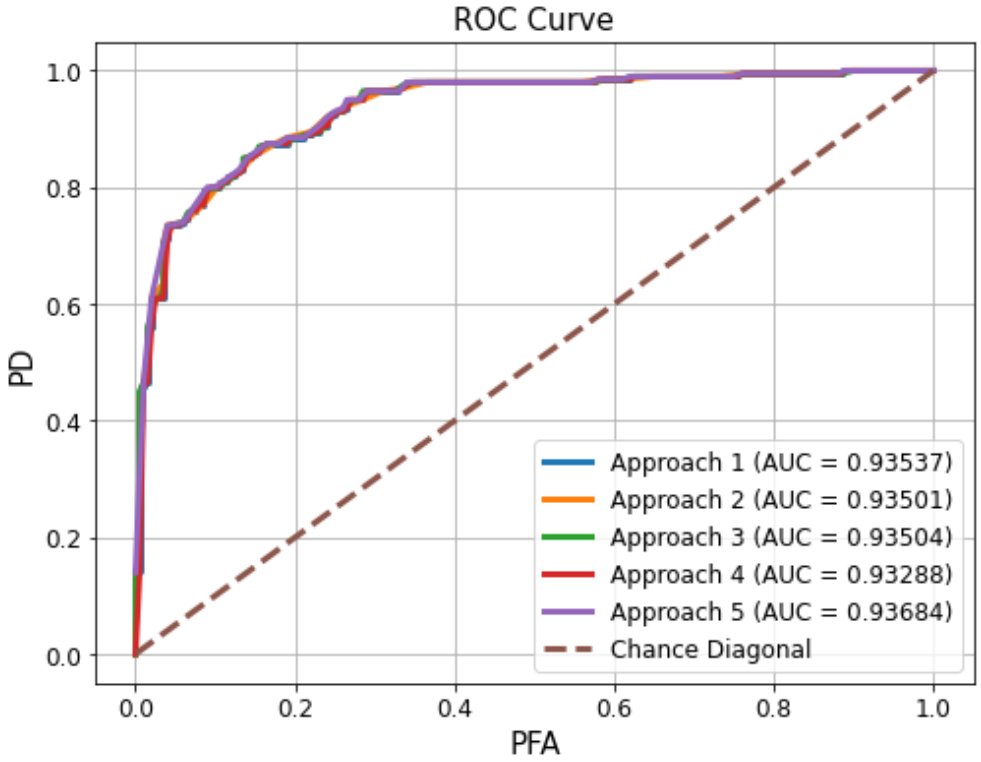
$$\lambda_{new} = [\lambda_{new-SmallerPart}, \lambda_{new-LargerPart}]$$

Question 3 (d) Solution:



Generating ROCs Section

Question 4 – Moderate data ROC curve (5 ROCs on the same set of axes) is shown below.



It should be noted that to better compare the 5 ROCs for each dataset in a quantitative method, I also calculate the area under curve (AUC) for each ROC curve and list the AUC values in the figure legend.

Question 4 (a) Solution:

For the moderate dataset, I think all the 5 approaches to selecting thresholds are appropriate, because based on the above plot, we can see that all 5 ROCs are very similar in terms of shape and relative position within the ROC axes. If I have to determine which one is the best, I will select Approach 5 as the most appropriate approach, because it has the largest AUC value.

Question 4 (b) Solution:

For the moderate dataset, I think none of the 5 approaches is inappropriate for the moderate dataset, because all 5 ROCs look very similar in terms of shape and relative position within the ROC axes, and they all have very similar and large AUC values.

My categorization for each of the 5 approaches for the moderate dataset is summarized below.

Approach 1 is appropriate for the moderate set of decision statistics.

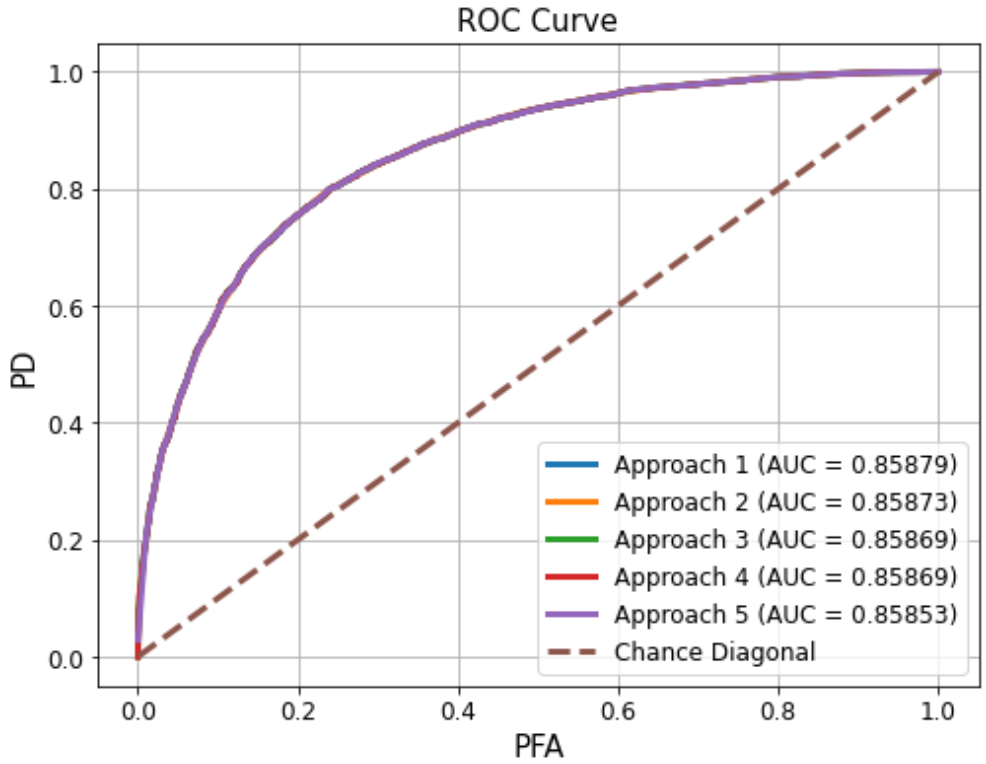
Approach 2 is appropriate for the moderate set of decision statistics.

Approach 3 is appropriate for the moderate set of decision statistics.

Approach 4 is appropriate for the moderate set of decision statistics.

Approach 5 is appropriate for the moderate set of decision statistics.

Question 5 – Big data ROC curve (5 ROCs on the same set of axes) is shown below.



Question 5 (a) – Solution:

For the big dataset, I think all the 5 approaches to selecting thresholds are appropriate, because based on the above plot, we can see that all 5 ROCs are very similar in terms of shape and relative position within the ROC axes. If I have to determine which one is the best, I will select Approach 1 as the most appropriate approach, because it has the largest AUC value.

Question 5 (b) – Solution:

For the big dataset, I think none of the 5 approaches is inappropriate for the big dataset, because all 5 ROCs look very similar in terms of shape and relatively position within the ROC axes, and they all have very similar and large AUC values.

My categorization for each of the 5 approaches for the big dataset is summarized below.

Approach 1 is appropriate for the big set of decision statistics.

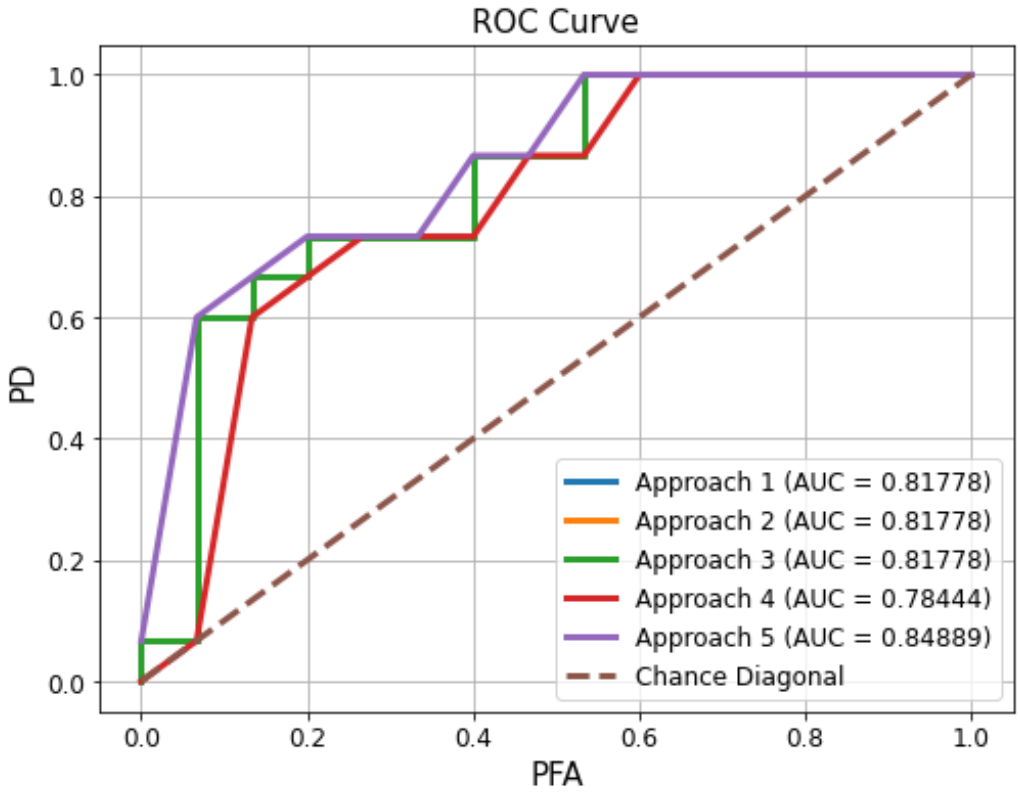
Approach 2 is appropriate for the big set of decision statistics.

Approach 3 is appropriate for the big set of decision statistics.

Approach 4 is appropriate for the big set of decision statistics.

Approach 5 is appropriate for the big set of decision statistics.

Question 6 – Small data ROC curve (5 ROCs on the same set of axes) is shown below.



Question 6 (a) Solution:

For the small dataset, I think Approaches 1, 2, 3, and 5 are appropriate because they all have AUC values greater than 0.80. In specific, I think Approach 5 is the most appropriate approach because it is above all the other 4 ROCs in terms of shape and relative position within the ROC axes, and it has the largest AUC value.

Question 6 (b) Solution:

For the small dataset, I think Approach 4 is inappropriate because its ROC is below all the other 4 ROCs in terms of shape and relative position within the ROC axes, and it has the lowest AUC value (0.784), which is the only AUC value below 0.80 among all 5 approaches.

My categorization for each of the 5 approaches for the small dataset is summarized below.

Approach 1 is appropriate for the small set of decision statistics.

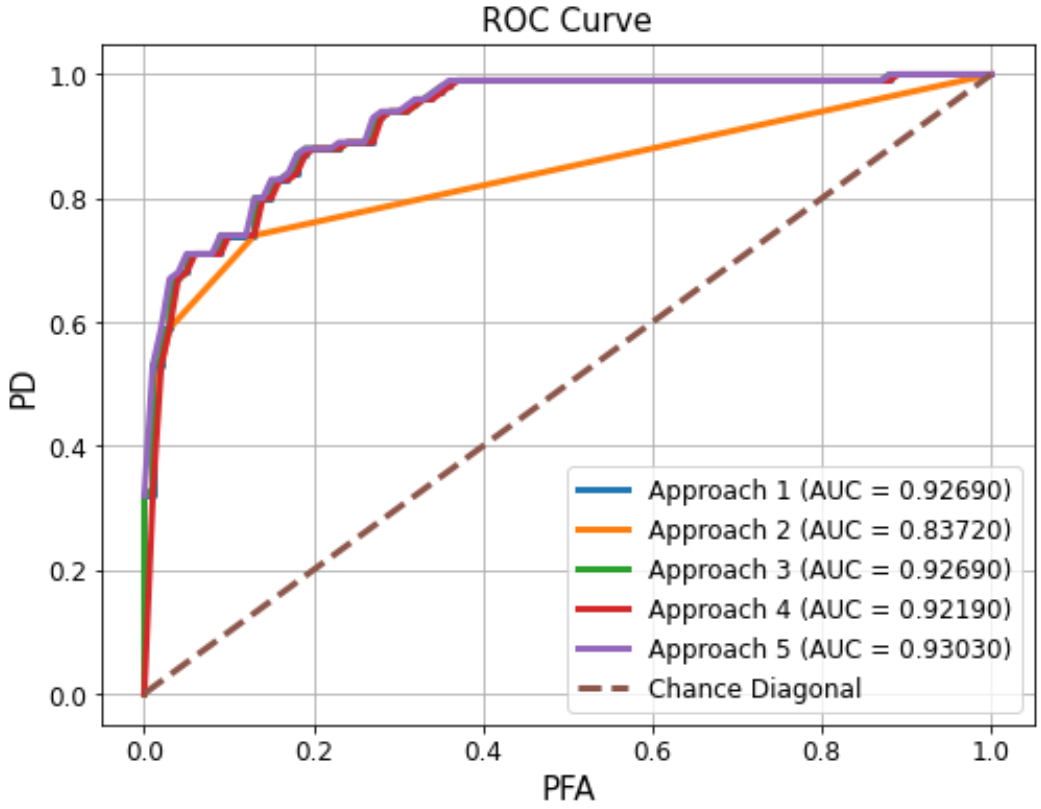
Approach 2 is appropriate for the small set of decision statistics.

Approach 3 is appropriate for the small set of decision statistics.

Approach 4 is inappropriate for the small set of decision statistics.

Approach 5 is appropriate for the small set of decision statistics.

Question 7 – Log-normal data ROC curve (5 ROCs on the same set of axes) is shown below.



Question 7 (a) Solution:

For the log-normal dataset, I think Approaches 1, 3, 4, and 5 are appropriate because they all have similar ROCs in terms of shape and relative position within the ROC axes, and they all have AUC values greater than 0.92. In specific, I think Approach 5 is the most appropriate approach, because its ROC has the largest AUC value.

Question 7 (b) Solution:

For the log-normal dataset, I think Approach 2 is inappropriate, because its ROC lies below the other 4 ROCs, and it has the lowest AUC value (0.837), which is the only AUC value below 0.9 among all 5 approaches.

My categorization for each of the 5 approaches for the log-normal dataset is summarized below.

Approach 1 is appropriate for the log normal set of decision statistics.

Approach 2 is inappropriate for the log normal set of decision statistics.

Approach 3 is appropriate for the log normal set of decision statistics.

Approach 4 is appropriate for the log normal set of decision statistics.

Approach 5 is appropriate for the log normal set of decision statistics.

Question 8 Solution:

Explanation 1 – Approach 1 is universally applicable, because when we are sweeping the threshold to generate the ROC curve, choosing every (n = 1) decision statistic will give us the finest resolution version of the ROC curve. Therefore, I think Approach 1 is universally applicable (meaning it will provide a good representation of the ROC without unnecessary computations), which has also been demonstrated by the previous 4 sets of decision statistics (moderate, big, small, and log-normal).

Explanation 2 – Approach 2 is not universally applicable, because choosing 99 linearly spaced thresholds from $\min(\lambda)$ to $\max(\lambda)$ could be a big problem for decision statistics which are not linearly distributed. Take the previous log-normal dataset as a concrete example, the decision statistics have log-normal distribution, and the quality of the ROC curve generated by Approach 2 is very bad, which means Approach 2 is inappropriate for the log-normal dataset. Therefore, I think Approach 2 is not universally applicable.

Explanation 3 – Approach 3 is universally applicable, because firstly, we have the flexibility to set n to 1 if there are too few decision statistics, which means we can achieve the finest resolution even with a very small dataset. Secondly, when choosing 99 decision statistics as thresholds, we have the flexibility to adjust the value of n, which helps this approach become more adaptable towards all types of decision statistics. Therefore, I think Approach 3 is universally applicable (meaning it will provide a good representation of the ROC without unnecessary computations), which has also been demonstrated by the previous 4 sets of decision statistics (moderate, big, small, and log-normal).

Explanation 4 – Approach 4 is not universally applicable, because although choosing every H_0 decision statistic as a threshold could help better interpret the probability of false alarm, there could be a big problem when we try to interpret the probability of detection from H_1 decision statistics, especially if we have a smaller-sized dataset. Take the previous small dataset as a concrete example, the quality of the ROC curve generated by Approach 4 is very bad. Therefore, I think Approach 4 is not universally applicable.

Explanation 5 – Approach 5 is universally applicable. To explain, if we can choose thresholds flexibly so that the probability of false alarm (PFA) is linearly sampled from 0 to 1 at an interval of 0.01, then the ROC curve can be represented with very good quality, because PFA directly represents the horizontal axis (x-axis) of ROC. If one axis of ROC becomes smoother or more detailed represented, then a good representation of the ROC curve can also be expected. Therefore, I think Approach 5 is universally applicable, which has also been demonstrated by the previous 4 sets of decision statistics (moderate, big, small, log-normal).

Question 9 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.
I use metrics from sklearn to calculate area under curve (AUC).

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

[2]:
```
file_moderate = "moderateData.csv"
file_big      = "bigData.csv"
file_small    = "smallData.csv"
file_log      = "logNormalData.csv"

df_moderate   = pd.read_csv(file_moderate, header = None)
df_big        = pd.read_csv(file_big,      header = None)
df_small      = pd.read_csv(file_small,    header = None)
df_log        = pd.read_csv(file_log,      header = None)
```

[3]:
```
arr_moderate = df_moderate.to_numpy()
arr_big      = df_big.to_numpy()
arr_small    = df_small.to_numpy()
arr_log      = df_log.to_numpy()
print(arr_moderate.shape)
print(arr_big.shape)
print(arr_small.shape)
print(arr_log.shape)
```

```
(400, 2)
(10000, 2)
(30, 2)
(200, 2)
```

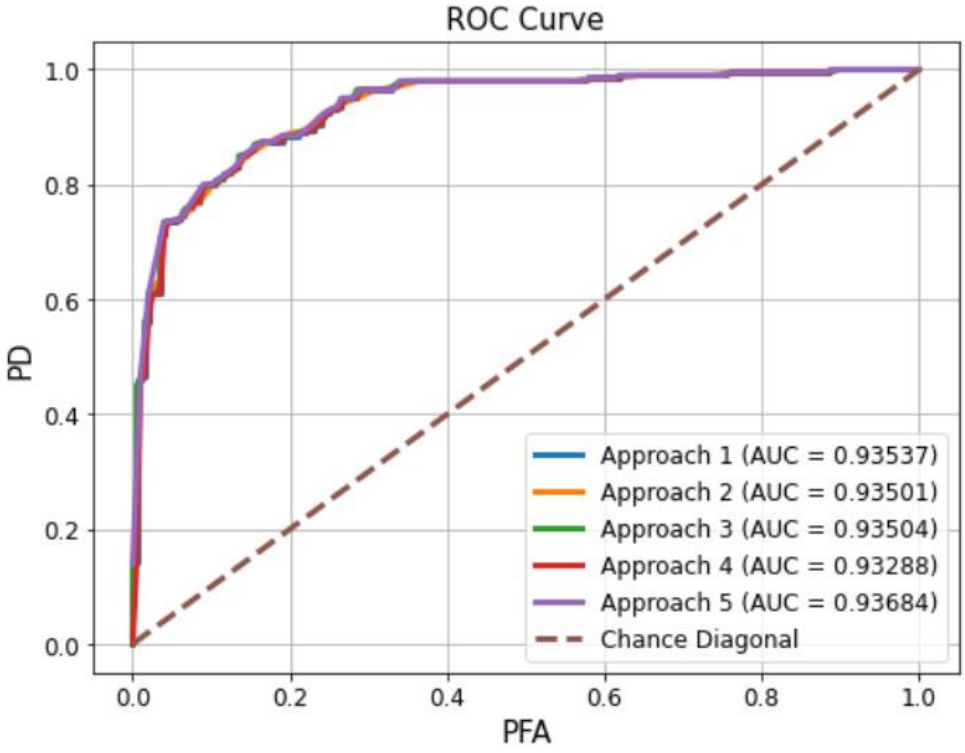
[4]:
```
def divide_labels (arr_data) :
    alldat = np.copy(arr_data)
    dat_zero = alldat[alldat[:,0]==0, :]
    dat_one = alldat[alldat[:,0]==1, :]
    return dat_zero, dat_one

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
```

[5]:
```
def flex_thresholds (arr_data, flex_type) :
    alldata = np.copy(arr_data)
    if flex_type == 1 :
        thres = np.zeros(2 + alldata.shape[0])
        thres[1:-1] = np.sort(alldata[:, 1])
        thres[0] = float("-inf")
        thres[-1] = float("inf")

    if flex_type == 2 :
        thres = np.zeros(101)
        lam_min = np.min(alldata[:, 1])
        lam_max = np.max(alldata[:, 1])
        thres[1:-1] = np.linspace(lam_min, lam_max, num = 99)
        thres[0] = float("-inf")
        thres[-1] = float("inf")

    if flex_type == 3 :
        if alldata.shape[0] <= 99 :
            # n = 1
            thres = np.zeros(2 + alldata.shape[0])
            thres[1:-1] = np.sort(alldata[:, 1])
        if alldata.shape[0] > 99 :
            # n is chosen by linear spacing
            thres = np.zeros(101)
            sortdata = np.sort(alldata[:, 1])
            indices = np.linspace(0, len(sortdata) - 1, num = 99).astype(int)
            thres[1:-1] = sortdata[indices]
        thres[0] = float("-inf")
        thres[-1] = float("inf")

    if flex_type == 4 :
        data_zero, data_one = divide_labels(alldata)
        thres = np.zeros(2 + data_zero.shape[0])
        thres[1:-1] = np.sort(data_zero[:, 1])
        thres[0] = float("-inf")
        thres[-1] = float("inf")

    if flex_type == 5 :
        data_zero, data_one = divide_labels(alldata)
        sort_zero = np.sort(data_zero[:, 1])
        step_size = 0.01
        sweep_thres = np.arange(sort_zero[0] - step_size, sort_zero[-1] + (2 * step_size), step_size)
        prev_thre = sweep_thres[0]
        desired_fas = np.linspace(0, 1, num = 101)
        thres = np.zeros(101)
        for i in range(101) :
            for curr_thre in sweep_thres[sweep_thres > prev_thre] :
                curr_fa = float(len(sort_zero[sort_zero >= curr_thre])) / len(sort_zero)
                if curr_fa <= desired_fas[101-1-i] and np.allclose(curr_fa, desired_fas[101-1-i],
                                                                   rtol = 0.005, atol = 0.005) :
                    thres[i] = curr_thre
                    prev_thre = curr_thre
                    break
                else :
                    thres[i] = prev_thre
    return thres
```

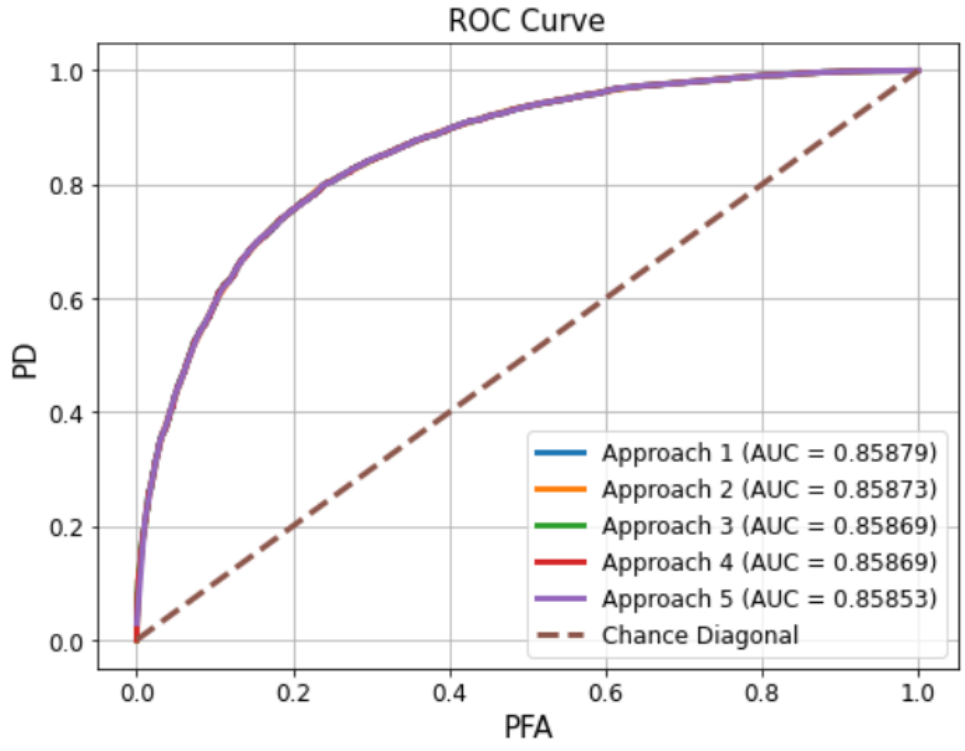
[6]:
```
def compute_ROC (arr_data, flex_type) :
    arr_thres = flex_thresholds(arr_data, flex_type)
    arr_PD = np.zeros(len(arr_thres))
    arr_PFA = np.zeros(len(arr_thres))
    H0, H1 = divide_labels(arr_data)
    for i in range(len(arr_thres)) :
        arr_PD[i] = compute_PD(H1, arr_thres[i])
        arr_PFA[i] = compute_PFA(H0, arr_thres[i])
    return np.flipud(arr_PD), np.flipud(arr_PFA)
```

[7]:
```
def plot_ROC (arr_data) :
    s1_pd, s1_pfa = compute_ROC(arr_data, flex_type = 1)
    s2_pd, s2_pfa = compute_ROC(arr_data, flex_type = 2)
    s3_pd, s3_pfa = compute_ROC(arr_data, flex_type = 3)
    s4_pd, s4_pfa = compute_ROC(arr_data, flex_type = 4)
    s5_pd, s5_pfa = compute_ROC(arr_data, flex_type = 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)
    refln = np.linspace(0, 1, num = 101)
    figure, axis = plt.subplots()
    axis.plot(s1_pfa, s1_pd, label = "Approach 1 (AUC = %.5f)" % auc1, linewidth = 3)
    axis.plot(s2_pfa, s2_pd, label = "Approach 2 (AUC = %.5f)" % auc2, linewidth = 3)
    axis.plot(s3_pfa, s3_pd, label = "Approach 3 (AUC = %.5f)" % auc3, linewidth = 3)
    axis.plot(s4_pfa, s4_pd, label = "Approach 4 (AUC = %.5f)" % auc4, linewidth = 3)
    axis.plot(s5_pfa, s5_pd, label = "Approach 5 (AUC = %.5f)" % auc5, 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
```

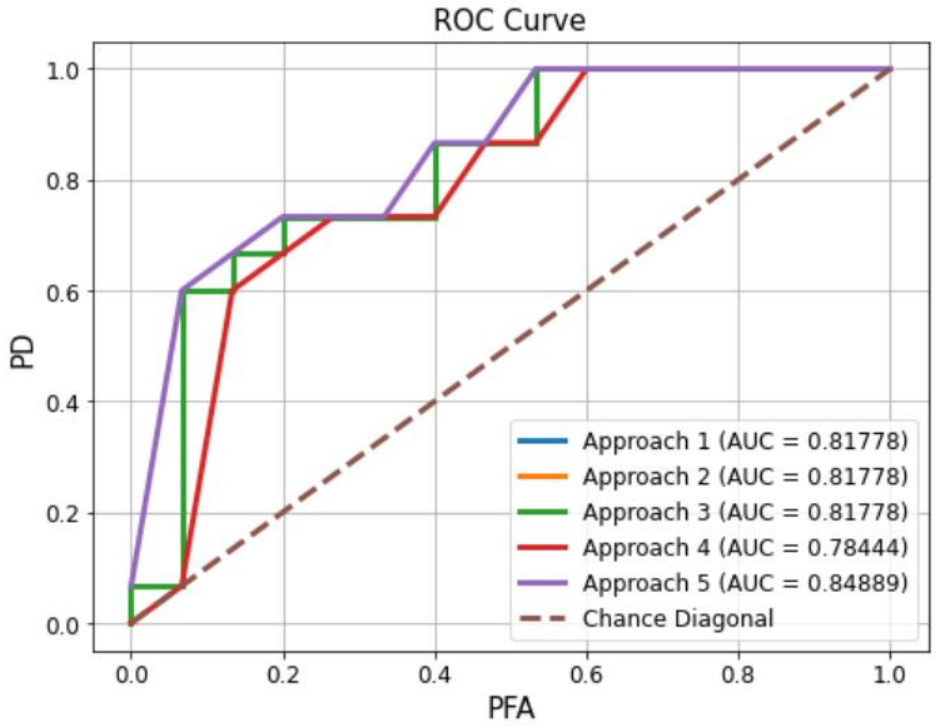
[8]:
```
plot_ROC(arr_moderate)
```



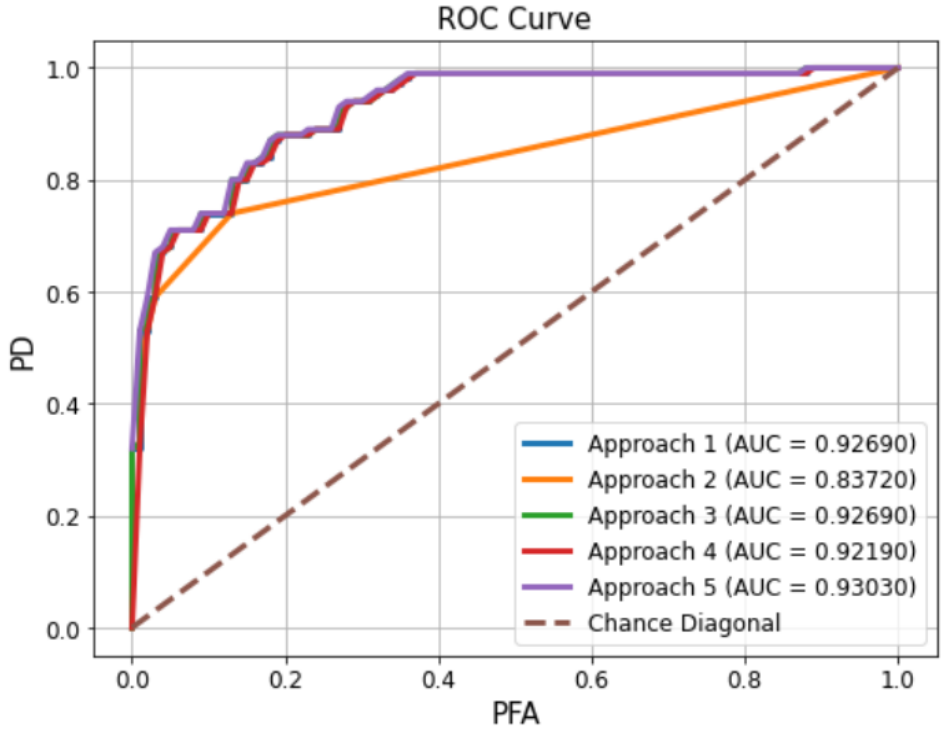
[9]:
```
plot_ROC(arr_big)
```



[10]:
```
plot_ROC(arr_small)
```



[11]:
```
plot_ROC(arr_log)
```



Probabilistic Decision Rules Section

Question 10 Solution:

Given two operating points $(P_{D1} = 1,\ \beta_1 = 0)$ and $(P_{D2} = 0.8,\ \beta_2 = 1/3)$ on the ROC, if we want to operate, on average, at $P_D = 0.95$, the probabilistic decision rule should be derived as shown below.

We assume a probability of p_1 for the occurrence of the first operating point, $(P_{D1} = 1,\ \beta_1 = 0)$.

We assume a probability of p_2 for the occurrence of the second operating point, $(P_{D2} = 0.8,\ \beta_2 = 1/3)$.

Since we want to choose an operating point on the straight line connecting two sampled operating points on the ROC, we can easily derive that $p_1 + p_2 = 1$.

Combining this condition with our expected operating point, $P_D = 0.95$, we can have the following equation set.

$$\begin{cases} p_1 + p_2 = 1 \\ p_1 P_{D1} + p_2 P_{D2} = 1p_1 + 0.8p_2 = P_D = 0.95 \end{cases}$$

Solving this equation set, we can get the following results.

$$\begin{cases} p_1 = 0.75 \\ p_2 = 0.25 \end{cases}$$

Therefore, the probability decision rule should be:

Give 75% probability for the first operating point, $(P_{D1} = 1,\ \beta_1 = 0)$.

Give 25% probability for the second operating point, $(P_{D2} = 0.8,\ \beta_2 = 1/3)$.

This probability decision rule will allow us to operate, on average, at $P_D = 0.95$.

Question 11 (a) Solution:

To concretely demonstrate that the expected value of P_D over many (100) simulations is indeed 0.95, here I will directly report the print-out results of my codes.

[9]:
```
# 0.75 * 1.0 + 0.25 * 0.8 = 0.95
arr_thres = np.array([0.,    0.33333])
arr_prob  = np.array([0.75,  0.25])
```

[10]:
```
sim_pd, sim_pfa = simulate_PDR(arr_knn, arr_thres, arr_prob,
                               nums = 100)
```

[11]:
```
print(sim_pd.shape)
print(sim_pfa.shape)
```

```
(100,)
(100,)
```

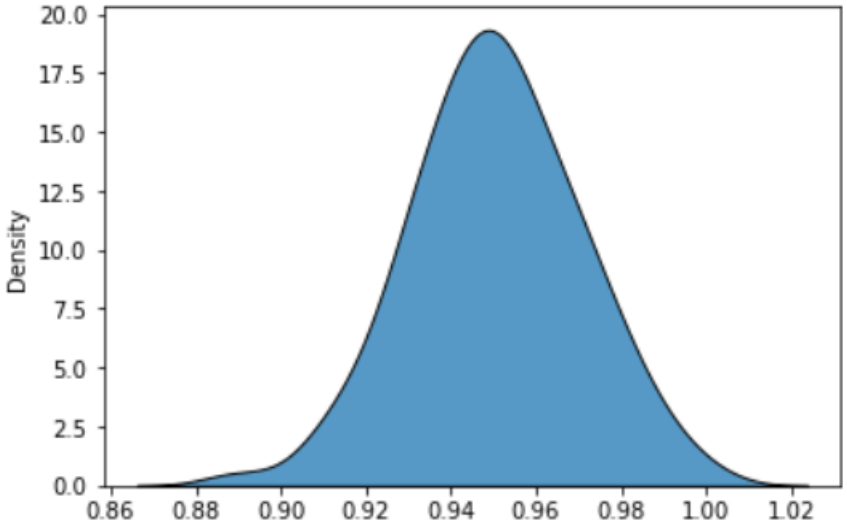
[12]:
```
avg_pd  = np.mean(sim_pd)
avg_pfa = np.mean(sim_pfa)
print("The Expected Value of PD  = %.2f" % avg_pd)
print("The Expected Value of PFA = %.2f" % avg_pfa)
```

```
The Expected Value of PD  = 0.95
The Expected Value of PFA = 0.82
```

Question 11 (a) Solution:

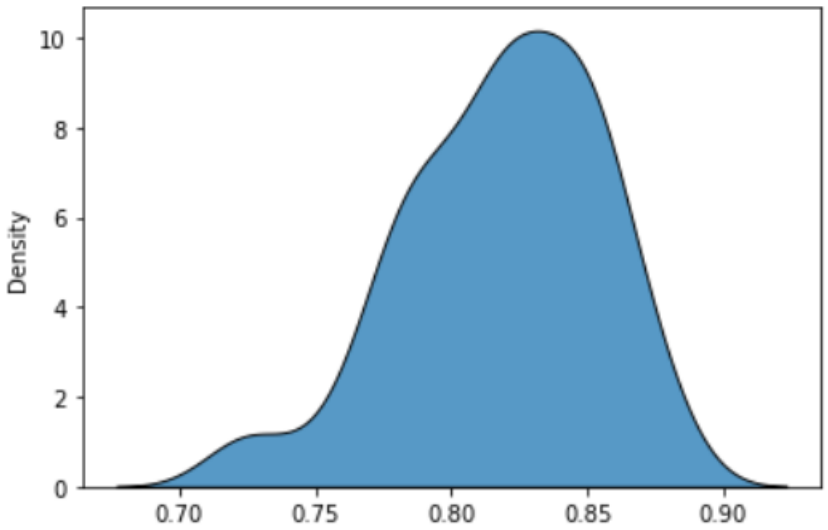
[13]:
```
avg_pd = np.mean(sim_pd)
print("The Expected Value of PD = %.2f" % avg_pd)
print("The probability density function of PD is shown below.")
sns.kdeplot(data = sim_pd, multiple = "stack");
```

```
The Expected Value of PD = 0.95
The probability density function of PD is shown below.
```



[14]:
```
avg_pfa = np.mean(sim_pfa)
print("The Expected Value of PFA = %.2f" % avg_pfa)
print("The probability density function of PFA is shown below.")
sns.kdeplot(data = sim_pfa, multiple = "stack");
```

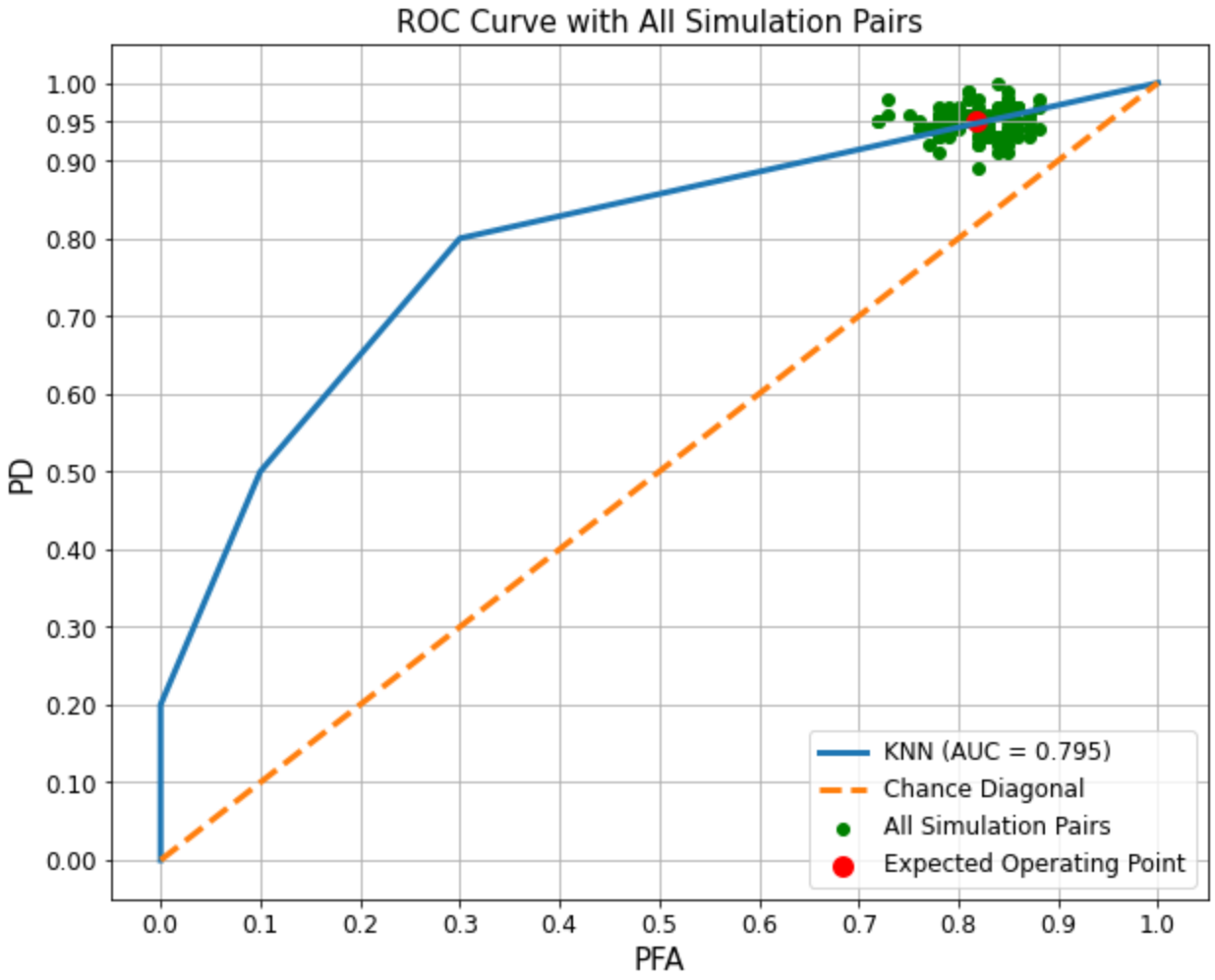
```
The Expected Value of PFA = 0.82
The probability density function of PFA is shown below.
```



Therefore, we can show that $\mathbb{E}\{\hat{P}_D\} = \mu_{\hat{P}_D} = 0.95$, and $\mathbb{E}\{\hat{P}_{FA}\} = \mu_{\hat{P}_{FA}} = 0.82$.

Question 11 (b) Solution:

The ROC plot with all the $(\hat{P}_D, \hat{P}_{FA})$ pairs produced by each individual simulation and the expected operating point $(\mu_{\hat{P}_{FA}}, \mu_{\hat{P}_D})$ from my probabilistic decision rule is shown below.



Question – Do the individual operating points cluster around the ROC in the vicinity of $P_D \approx 0.95$*?*

Answer – Yes. Based on all the green points in the above plot, we can demonstrate that the individual operating points indeed cluster around the ROC in the vicinity of $P_D \approx 0.95$.

Question – Does your expected operating point have $P_D \approx 0.95$ *and fall on (or very near to) the line connecting the* $\beta = 1/3$ *and* $\beta = 0$ *operating points?*

Answer – Yes. Based on the red point in the above plot, we can demonstrate that the expected operating point indeed has $P_D \approx 0.95$ and falls on the line connecting the $\beta = 1/3$ and $\beta = 0$ operating points.

Question 12 Solution:

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

Reference the Packages/Toolboxes

I use numpy to process arraies and matrices.
I also use numpy.random.choice to implement my probability decision rule.
I use pandas to read and process the original dataset.
I use seaborn to implement kernel density estimation and
to estimate the probability density functions of PD and PFA.
I use matplotlib.pyplot to plot ROC.
I use metrics from sklearn to calculate area under curve (AUC).

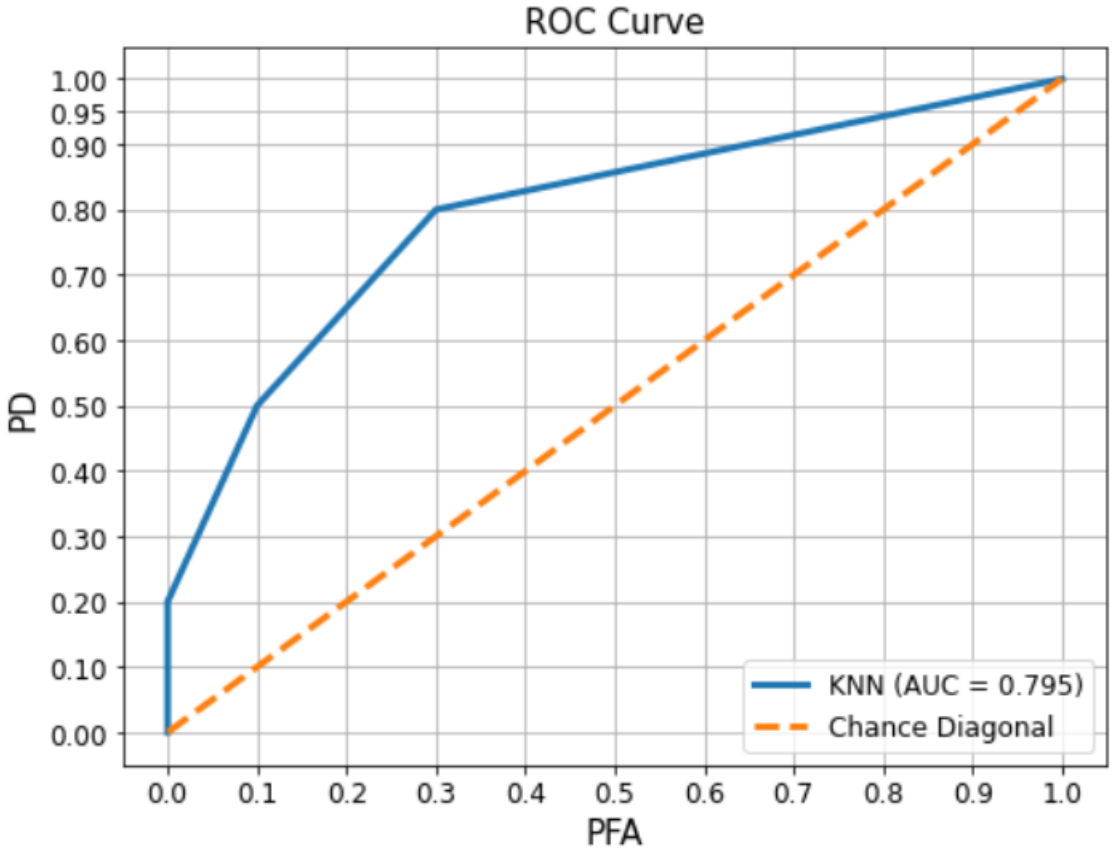
[2]:
```
file_knn = "knn3DecisionStatistics.csv"
df_knn = pd.read_csv(file_knn, header = None)
arr_knn = df_knn.to_numpy()
arr_beta = np.zeros(3 + 2)
arr_beta[0] = 0.
arr_beta[1] = 0.33333
arr_beta[2] = 0.66667
arr_beta[3] = 1.
arr_beta[4] = float("inf")
print(arr_knn.shape)
```

```
(200, 2)
```

[3]:
```
def divide_labels (arr_data) :
    alldat = np.copy(arr_data)
    dat_zero = alldat[alldat[:,0]==0, :]
    dat_one = alldat[alldat[:,0]==1, :]
    return dat_zero, dat_one
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
def compute_ROC (arr_data, arr_thres) :
    arr_PD = np.zeros(len(arr_thres))
    arr_PFA = np.zeros(len(arr_thres))
    H0, H1 = divide_labels(arr_data)
    for i in range(len(arr_thres)) :
        arr_PD[i] = compute_PD(H1, arr_thres[i])
        arr_PFA[i] = compute_PFA(H0, arr_thres[i])
    return np.flipud(arr_PD), np.flipud(arr_PFA)
```

[4]:
```
def plot_ROC (arr_data, arr_thres) :
    sl_pd, sl_pfa = compute_ROC(arr_data, arr_thres)
    auc1 = metrics.auc(sl_pfa, sl_pd)
    refln = np.linspace(0, 1, num = 101)
    figure, axis = plt.subplots()
    axis.plot(sl_pfa, sl_pd, label = "KNN (AUC = %.3f)" % auc1, 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)
    x_tick = np.linspace(0, 1, num = 11)
    y_tick = np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1.0])
    plt.xticks(x_tick, fontsize = 12)
    plt.yticks(y_tick, fontsize = 12)
    plt.legend(loc = "lower right", fontsize = 12)
    plt.show()
    return None
```

[5]:
```
plot_ROC(arr_knn, arr_beta)
```



[6]:
```
# 0.75 * 1.0 + 0.25 * 0.8 = 0.95
arr_thres = np.array([0.,    0.33333])
arr_prob  = np.array([0.75,  0.25])
```

[7]:
```
def compute_prob_PD (H1, arr_thres, arr_prob) :
    prob_thres = np.random.choice(arr_thres, size = H1.shape[0], p = arr_prob)
    countd = 0
    totalh = H1.shape[0]
    for i in range(totalh) :
        if H1[i, 1] >= prob_thres[i] :
            countd = countd + 1
    detection = float(countd) / totalh
    return detection
def compute_prob_PFA (H0, arr_thres, arr_prob) :
    prob_thres = np.random.choice(arr_thres, size = H0.shape[0], p = arr_prob)
    countfa = 0
    totalh = H0.shape[0]
    for i in range(totalh) :
        if H0[i, 1] >= prob_thres[i] :
            countfa = countfa + 1
    false_alarm = float(countfa) / totalh
    return false_alarm
```

[8]:
```
def simulate_PDR (arr_data, arr_thres, arr_prob, nums = 100) :
    simpd = np.zeros(nums)
    simpfa = np.zeros(nums)
    for i in range(nums) :
        H0, H1 = divide_labels(arr_data)
        simpd[i] = compute_prob_PD(H1, arr_thres, arr_prob)
        simpfa[i] = compute_prob_PFA(H0, arr_thres, arr_prob)
    return simpd, simpfa
```

[9]:
```
# 0.75 * 1.0 + 0.25 * 0.8 = 0.95
arr_thres = np.array([0.,    0.33333])
arr_prob  = np.array([0.75,  0.25])
```

[10]:
```
sim_pd, sim_pfa = simulate_PDR(arr_knn, arr_thres, arr_prob,
                               nums = 100)
```

[11]:
```
print(sim_pd.shape)
print(sim_pfa.shape)
```

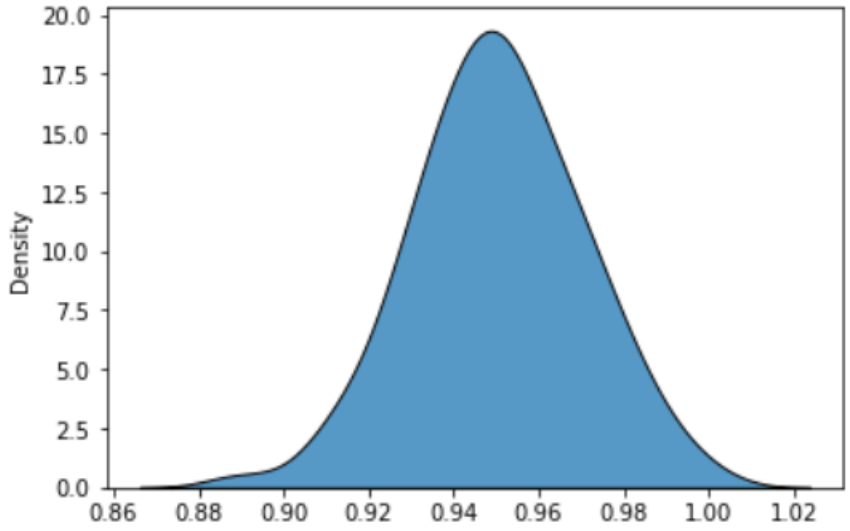
```
(100,)
(100,)
```

[12]:
```
avg_pd  = np.mean(sim_pd)
avg_pfa = np.mean(sim_pfa)
print("The Expected Value of PD  = %.2f" % avg_pd)
print("The Expected Value of PFA = %.2f" % avg_pfa)
```

```
The Expected Value of PD  = 0.95
The Expected Value of PFA = 0.82
```

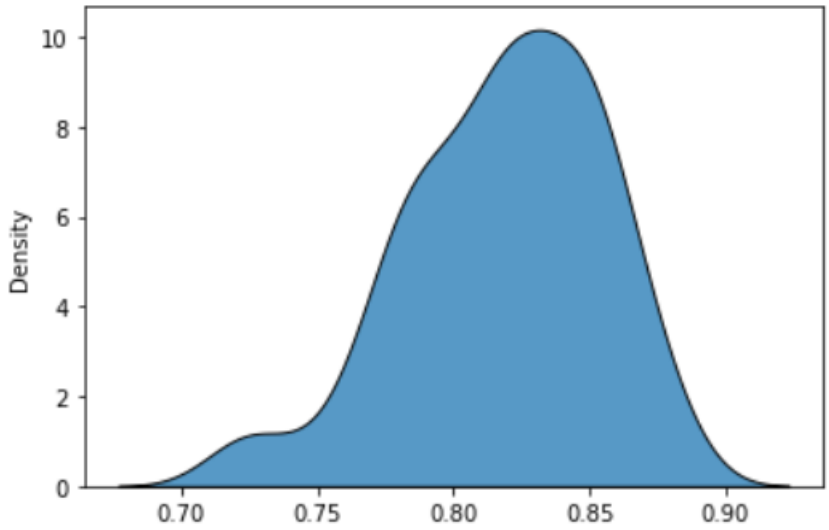
[13]:
```
avg_pd = np.mean(sim_pd)
print("The Expected Value of PD = %.2f" % avg_pd)
print("The probability density function of PD is shown below.")
sns.kdeplot(data = sim_pd, multiple = "stack");
```

```
The Expected Value of PD = 0.95
The probability density function of PD is shown below.
```



[14]:
```
avg_pfa = np.mean(sim_pfa)
print("The Expected Value of PFA = %.2f" % avg_pfa)
print("The probability density function of PFA is shown below.")
sns.kdeplot(data = sim_pfa, multiple = "stack");
```

```
The Expected Value of PFA = 0.82
The probability density function of PFA is shown below.
```



```python
def plot_ROC_simulations (arr_data, arr_thres, simu_pd, simu_pfa) :
    s1_pd, s1_pfa = compute_ROC(arr_data, arr_thres)
    auc1 = metrics.auc(s1_pfa, s1_pd)
    refln = np.linspace(0, 1, num = 101)
    avg_pd = np.mean(simu_pd)
    avg_pfa = np.mean(simu_pfa)
    figure, axis = plt.subplots()
    axis.plot(s1_pfa, s1_pd, label = "KNN (AUC = %.3f)" % auc1, linewidth = 3)
    axis.plot(refln, refln, "--", label = "Chance Diagonal", linewidth = 3)
    axis.scatter(simu_pfa, simu_pd, c = "g", label = "All Simulation Pairs", linewidths = 1)
    axis.scatter(avg_pfa, avg_pd, c = "r", label = "Expected Operating Point", linewidths = 5)
    axis.set_xlabel("PFA", fontsize = 15)
    axis.set_ylabel("PD", fontsize = 15)
    axis.set_title("ROC Curve with All Simulation Pairs", fontsize = 15)
    axis.grid()
    figure.set_size_inches(10, 8)
    x_tick = np.linspace(0, 1, num = 11)
    y_tick = np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1.0])
    plt.xticks(x_tick, fontsize = 12)
    plt.yticks(y_tick, fontsize = 12)
    plt.legend(loc = "lower right", fontsize = 12)
    plt.show()
    return None
```

```python
plot_ROC_simulations(arr_knn, arr_beta, sim_pd, sim_pfa)
```