

Project Task 4

Hrishikesh Deshpande (hd11) - Task 4.1

Karthik Appana (kappana2) - Task 4.2

Siddharth Gummadapu (sg96) - Task 4.3 & Task 3 Recap

```
In [37]: # imports
import numpy as np
import scipy.io as sio
import scipy.stats as stat
import matplotlib.pyplot as plt
```

Task 3 Recap

Patient 1

```
In [38]: # load data from mat file
pat1 = sio.loadmat('data_patients/patient1.mat')

pat1Floor = np.floor(pat1['all_data'])

pat1Cutoff = int(len(pat1['all_data'])[0]) * (2/3))

pat1TrainData = []
for i in range(7):
    pat1TrainData.append(pat1Floor[i][:pat1Cutoff])
pat1TrainLabels = pat1['all_labels'][0][:pat1Cutoff]

pat1TestData = []
for i in range(7):
    pat1TestData.append(pat1Floor[i][pat1Cutoff:])
pat1TestLabels = pat1['all_labels'][0][pat1Cutoff:]
```

Patient 2

```
In [39]: # load data from mat file
pat2 = sio.loadmat('data_patients/patient2.mat')

pat2Floor = np.floor(pat2['all_data'])

pat2Cutoff = int(len(pat2['all_data'][0]) * (2/3))

pat2TrainData = []
for i in range(7):
    pat2TrainData.append(pat2Floor[i][:pat2Cutoff])
pat2TrainLabels = pat2['all_labels'][0][:pat2Cutoff]

pat2TestData = []
for i in range(7):
    pat2TestData.append(pat2Floor[i][pat2Cutoff:])
pat2TestLabels = pat2['all_labels'][0][pat2Cutoff:]
```

Patient 3

```
In [40]: # load data from mat file
pat3 = sio.loadmat('data_patients/patient3.mat')

pat3Floor = np.floor(pat3['all_data'])

pat3Cutoff = int(len(pat3['all_data'][0]) * (2/3))

pat3TrainData = []
for i in range(7):
    pat3TrainData.append(pat3Floor[i][:pat3Cutoff])
pat3TrainLabels = pat3['all_labels'][0][:pat3Cutoff]

pat3TestData = []
for i in range(7):
    pat3TestData.append(pat3Floor[i][pat3Cutoff:])
pat3TestLabels = pat3['all_labels'][0][pat3Cutoff:]
```

Patient 4

```
In [41]: pat4 = sio.loadmat('data_patients/patient4.mat')

pat4Floor = np.floor(pat4['all_data'])
```

```

pat4Cutoff = int(len(pat4['all_data'][0]) * (2/3))

pat4TrainData = []
for i in range(7):
    pat4TrainData.append(pat4Floor[i][:pat4Cutoff])
pat4TrainLabels = pat4['all_labels'][0][:pat4Cutoff]

pat4TestData = []
for i in range(7):
    pat4TestData.append(pat4Floor[i][pat4Cutoff:])
pat4TestLabels = pat4['all_labels'][0][pat4Cutoff:]

```

Patient 5

```

In [42]: pat5 = sio.loadmat('data_patients/patient5.mat')

pat5Floor = np.floor(pat5['all_data'])

pat5Cutoff = int(len(pat5['all_data'][0]) * (2/3))

pat5TrainData = []
for i in range(7):
    pat5TrainData.append(pat5Floor[i][:pat5Cutoff])
pat5TrainLabels = pat5['all_labels'][0][:pat5Cutoff]

pat5TestData = []
for i in range(7):
    pat5TestData.append(pat5Floor[i][pat5Cutoff:])
pat5TestLabels = pat5['all_labels'][0][pat5Cutoff:]

```

Patient 6

```

In [43]: pat6 = sio.loadmat('data_patients/patient6.mat')

pat6Floor = np.floor(pat6['all_data'])

pat6Cutoff = int(len(pat6['all_data'][0]) * (2/3))

pat6TrainData = []
for i in range(7):
    pat6TrainData.append(pat6Floor[i][:pat6Cutoff])
pat6TrainLabels = pat6['all_labels'][0][:pat6Cutoff]

```

```
pat6TestData = []
for i in range(7):
    pat6TestData.append(pat6Floor[i][pat6Cutoff:])
pat6TestLabels = pat6['all_labels'][0][pat6Cutoff:]
```

Patient 7

```
In [44]: pat7 = sio.loadmat('data_patients/patient7.mat')

pat7Floor = np.floor(pat7['all_data'])

pat7Cutoff = int(len(pat7['all_data'][0]) * (2/3))

pat7TrainData = []
for i in range(7):
    pat7TrainData.append(pat7Floor[i][:pat7Cutoff])
pat7TrainLabels = pat7['all_labels'][0][:pat7Cutoff]

pat7TestData = []
for i in range(7):
    pat7TestData.append(pat7Floor[i][pat7Cutoff:])
pat7TestLabels = pat7['all_labels'][0][pat7Cutoff:]
```

Patient 8

```
In [45]: pat8 = sio.loadmat('data_patients/patient8.mat')

pat8Floor = np.floor(pat8['all_data'])

pat8Cutoff = int(len(pat8['all_data'][0]) * (2/3))

pat8TrainData = []
for i in range(7):
    pat8TrainData.append(pat8Floor[i][:pat8Cutoff])
pat8TrainLabels = pat8['all_labels'][0][:pat8Cutoff]

pat8TestData = []
for i in range(7):
    pat8TestData.append(pat8Floor[i][pat8Cutoff:])
pat8TestLabels = pat8['all_labels'][0][pat8Cutoff:]
```

Patient 9

```
In [46]: pat9 = sio.loadmat('data_patients/patient9.mat')

pat9Floor = np.floor(pat9['all_data'])

pat9Cutoff = int(len(pat9['all_data'][0]) * (2/3))

pat9TrainData = []
for i in range(7):
    pat9TrainData.append(pat9Floor[i][:pat9Cutoff])
pat9TrainLabels = pat9['all_labels'][0][:pat9Cutoff]

pat9TestData = []
for i in range(7):
    pat9TestData.append(pat9Floor[i][pat9Cutoff:])
pat9TestLabels = pat9['all_labels'][0][pat9Cutoff:]
```

Prior Probabilities

```
In [47]: def getPriorVals(trainLabels):
    priorH0 = list(trainLabels).count(0) / len(trainLabels)
    priorH1 = list(trainLabels).count(1) / len(trainLabels)

    return round(priorH0, 2), round(priorH1, 2)
```

```
In [48]: pat1PriorH0, pat1PriorH1 = getPriorVals(pat1TrainLabels)
pat2PriorH0, pat2PriorH1 = getPriorVals(pat2TrainLabels)
pat3PriorH0, pat3PriorH1 = getPriorVals(pat3TrainLabels)
pat4PriorH0, pat4PriorH1 = getPriorVals(pat4TrainLabels)
pat5PriorH0, pat5PriorH1 = getPriorVals(pat5TrainLabels)
pat6PriorH0, pat6PriorH1 = getPriorVals(pat6TrainLabels)
pat7PriorH0, pat7PriorH1 = getPriorVals(pat7TrainLabels)
pat8PriorH0, pat8PriorH1 = getPriorVals(pat8TrainLabels)
pat9PriorH0, pat9PriorH1 = getPriorVals(pat9TrainLabels)
```

Calculating Likelihood Matrices

```
In [49]: def likelihoodMatrix(trainData, trainLabels):
    likelihoodMatrices = []
```

```

for featureIdx in range(7):
    featureData = trainData[featureIdx]
    labels = trainLabels

    minVal = int(np.min(featureData))
    maxVal = int(np.max(featureData))
    possibleValues = range(minVal, maxVal + 1)

    countMatrix = np.zeros((2, len(possibleValues)))

    for i in range(len(featureData)):
        classIdx = int(labels[i])
        valueIdx = int(featureData[i] - minVal)
        countMatrix[classIdx][valueIdx] += 1

    h0Total = np.sum(countMatrix[0])
    h1Total = np.sum(countMatrix[1])

    likelihoodMatrix = np.zeros((2, len(possibleValues)))

    for valIdx in range(len(possibleValues)):
        likelihoodMatrix[0][valIdx] = countMatrix[0][valIdx] / h0Total if h0Total > 0 else 0
        likelihoodMatrix[1][valIdx] = countMatrix[1][valIdx] / h1Total if h1Total > 0 else 0

    likelihoodMatrices.append(likelihoodMatrix)

return likelihoodMatrices

```

```

In [50]: pat1LM = likelihoodMatrix(pat1TrainData, pat1TrainLabels)
pat2LM = likelihoodMatrix(pat2TrainData, pat2TrainLabels)
pat3LM = likelihoodMatrix(pat3TrainData, pat3TrainLabels)
pat4LM = likelihoodMatrix(pat4TrainData, pat4TrainLabels)
pat5LM = likelihoodMatrix(pat5TrainData, pat5TrainLabels)
pat6LM = likelihoodMatrix(pat6TrainData, pat6TrainLabels)
pat7LM = likelihoodMatrix(pat7TrainData, pat7TrainLabels)
pat8LM = likelihoodMatrix(pat8TrainData, pat8TrainLabels)
pat9LM = likelihoodMatrix(pat9TrainData, pat9TrainLabels)

```

```

In [51]: def mlRule(likelihoodMatrices):
    MLArr = []
    for feature in likelihoodMatrices:
        featureMLArr = []
        for i in range(len(feature[0])):
            if feature[1][i] >= feature[0][i]:
                featureMLArr.append(1)

```

```

        else:
            featureMLArr.append(0)
        MLArr.append(featureMLArr)

    return MLArr

def mapRule(likelihoodMatrices, priorH0, priorH1):
    MAPArr = []
    for feature in likelihoodMatrices:
        featureMAPArr = []
        for i in range(len(feature[0])):
            if (priorH1 * feature[1][i]) >= (priorH0 * feature[0][i]):
                featureMAPArr.append(1)
            else:
                featureMAPArr.append(0)
        MAPArr.append(featureMAPArr)

    return MAPArr

```

Hypotheses Table

In [52]: `def hypTable(likelihoodMatrices, trainData, priorH0, priorH1):`

```

    mainTable = []
    for featIdx in range(7):
        table = []
        featureData = trainData[featIdx]
        minVal = int(np.min(featureData))
        maxVal = int(np.max(featureData))
        possibleValues = range(minVal, maxVal + 1)
        table.append([i for i in possibleValues])
        table.append(likelihoodMatrices[featIdx][0])
        table.append(likelihoodMatrices[featIdx][1])
        mlArr = mlRule(likelihoodMatrices)
        table.append(mlArr[featIdx])
        mapArr = mapRule(likelihoodMatrices, priorH0, priorH1)
        table.append(mapArr[featIdx])
        npTable = np.array(table)
        mainTable.append(npTable.T)

    return mainTable

```

```
In [53]: pat1Table = hypTable(pat1LM, pat1TrainData, pat1PriorH0, pat1PriorH1)
pat2Table = hypTable(pat2LM, pat2TrainData, pat2PriorH0, pat2PriorH1)
pat3Table = hypTable(pat3LM, pat3TrainData, pat3PriorH0, pat3PriorH1)
pat4Table = hypTable(pat4LM, pat4TrainData, pat4PriorH0, pat4PriorH1)
pat5Table = hypTable(pat5LM, pat5TrainData, pat5PriorH0, pat5PriorH1)
pat6Table = hypTable(pat6LM, pat6TrainData, pat6PriorH0, pat6PriorH1)
pat7Table = hypTable(pat7LM, pat7TrainData, pat7PriorH0, pat7PriorH1)
pat8Table = hypTable(pat8LM, pat8TrainData, pat8PriorH0, pat8PriorH1)
pat9Table = hypTable(pat9LM, pat9TrainData, pat9PriorH0, pat9PriorH1)

finalHypothesisTable = [pat1Table, pat2Table, pat3Table, pat4Table,
                        pat5Table, pat6Table, pat7Table, pat8Table, pat9Table]
```

Pair Decisions Based on Data

ML Rule: Patients 1, 3, 4

Patient 1 - Features 1 and 3:

In both the lowest error test and the golden alarm correlation test, both features 1 and 3 seemed to be the most accurate and prominent in determining the result for Patient 1.

Patient 3 - Features 1 and 7:

In both the lowest error test and the golden alarm correlation test, feature 1 seemed to be the most accurate and prominent in determining the result for Patient 3. The next step was to choose between lower error (feature 5) or higher impact (feature 7). In this case we decided to go with feature 7 as our second option because it also had a decently low error while providing a more significant impact on the patient's results.

Patient 4 - Features 2 and 5:

In both the lowest error test and the golden alarm correlation test, feature 5 seemed to be the most accurate and prominent in determining the result for Patient 4. The next step was to choose between lower error (feature 7) or higher impact (feature 2). In this case we decided to go with feature 2 as our second option because it had a decently low error while providing a much more significant impact on the patient's results as compared to feature 7.

MAP Rule: Patients 3, 5, 6

Patient 3 - Features 1 and 7:

In both the lowest error test and the golden alarm correlation test, feature 1 seemed to be the most accurate and prominent in determining the result for Patient 3. The next step was to choose between lower error (feature 5) or higher impact (feature 7). In this case we decided to go with feature 7 as our second option because it also had a decently low error while providing a more significant impact on the patient's results.

Patient 5 - Features 3 and 1:

For this patient, we got completely different pairs in both the lowest error test and the golden alarm correlation test. So we decided to choose the best of the 2 pairs and combine them into the pair of features for this patient. From the lower error test, we chose feature 1 since it had the lowest error, and from the golden alarm correlation test we chose feature 3 since it had the highest impact on determining the patient's results.

Patient 6 - Features 6 and 4:

In both the lowest error test and the golden alarm correlation test, feature 6 seemed to be the most accurate and prominent in determining the result for this patient. The next step was to choose between lower error (feature 3) or higher impact (feature 4). In this case we decided to go with feature 4 as our second option because it had a decently low error while providing a much more significant impact on the patient's results as compared to feature 3.

Task 4.1a - Joint Likelihood Matrices

```
In [54]: def jointLikelihoodMatrix(HT_table, featureIdx1, featureIdx2):
    table1 = HT_table[featureIdx1]
    table2 = HT_table[featureIdx2]

    values1 = table1[:, 0]
    values2 = table2[:, 0]

    likelihoodH0_1 = table1[:, 1]
    likelihoodH1_1 = table1[:, 2]

    likelihoodH0_2 = table2[:, 1]
    likelihoodH1_2 = table2[:, 2]

    jointH0 = np.outer(likelihoodH0_1, likelihoodH0_2)
    jointH1 = np.outer(likelihoodH1_1, likelihoodH1_2)

    return jointH0, jointH1, values1, values2
```

```
In [55]: pat1jointH0, pat1jointH1, pat1jointValues1, pat1jointValues2 = jointLikelihoodMatrix(pat1Table, 0, 2)
pat3jointH0, pat3jointH1, pat3jointValues1, pat3jointValues2 = jointLikelihoodMatrix(pat3Table, 0, 6)
pat4jointH0, pat4jointH1, pat4jointValues1, pat4jointValues2 = jointLikelihoodMatrix(pat4Table, 1, 4)
pat5jointH0, pat5jointH1, pat5jointValues1, pat5jointValues2 = jointLikelihoodMatrix(pat5Table, 0, 2)
pat6jointH0, pat6jointH1, pat6jointValues1, pat6jointValues2 = jointLikelihoodMatrix(pat6Table, 3, 5)
```

Task 4.1b - Joint ML and MAP Rule

```
In [56]: def jointMLRule(jointH0, jointH1):
    mlArr = np.zeros(jointH0.shape, dtype=int)
    mlArr[jointH1 >= jointH0] = 1
    mlArr[jointH1 < jointH0] = 0
    return mlArr

def jointMAPRule(jointH0, jointH1, priorH0, priorH1):
    mlScaledH0 = priorH0 * jointH0
    mlScaledH1 = priorH1 * jointH1
    mapArr = np.zeros(jointH0.shape, dtype=int)
    mapArr[mlScaledH1 >= mlScaledH0] = 1
    mapArr[mlScaledH1 < mlScaledH0] = 0
    return mapArr
```

```
In [57]: pat1jointML = jointMLRule(pat1jointH0, pat1jointH1)
pat1jointMAP = jointMAPRule(pat1jointH0, pat1jointH1, pat1PriorH0, pat1PriorH1)
```

```

pat3jointML = jointMLRule(pat3jointH0, pat3jointH1)
pat3jointMAP = jointMAPRule(pat3jointH0, pat3jointH1, pat3PriorH0, pat3PriorH1)

pat4jointML = jointMLRule(pat4jointH0, pat4jointH1)
pat4jointMAP = jointMAPRule(pat4jointH0, pat4jointH1, pat4PriorH0, pat4PriorH1)

pat5jointML = jointMLRule(pat5jointH0, pat5jointH1)
pat5jointMAP = jointMAPRule(pat5jointH0, pat5jointH1, pat5PriorH0, pat5PriorH1)

pat6jointML = jointMLRule(pat6jointH0, pat6jointH1)
pat6jointMAP = jointMAPRule(pat6jointH0, pat6jointH1, pat6PriorH0, pat6PriorH1)

```

Task 4.1c - Joint Hypothesis Table

```

In [58]: def jointHypTable(jointH0, jointH1, jointML, jointMAP, values1, values2):
    rows = []

    for i, val1 in enumerate(values1):
        for j, val2 in enumerate(values2):
            row = [
                val1,
                val2,
                jointH1[i, j],
                jointH0[i, j],
                jointML[i, j],
                jointMAP[i, j]
            ]
            rows.append(row)

    return np.array(rows)

```

```

In [59]: pat1JointHT = jointHypTable(pat1jointH0, pat1jointH1, pat1jointML, pat1jointMAP, pat1jointValues1, pat1jointValues2)
pat3JointHT = jointHypTable(pat3jointH0, pat3jointH1, pat3jointML, pat3jointMAP, pat3jointValues1, pat3jointValues2)
pat4JointHT = jointHypTable(pat4jointH0, pat4jointH1, pat4jointML, pat4jointMAP, pat4jointValues1, pat4jointValues2)
pat5JointHT = jointHypTable(pat5jointH0, pat5jointH1, pat5jointML, pat5jointMAP, pat5jointValues1, pat5jointValues2)
pat6JointHT = jointHypTable(pat6jointH0, pat6jointH1, pat6jointML, pat6jointMAP, pat6jointValues1, pat6jointValues2)

finalJointHypTables = [pat1JointHT, pat3JointHT, pat4JointHT, pat5JointHT, pat6JointHT]

```

Task 4.1d - Plotting Conditional Joint PDFS

From this point onward, we will be narrowing down to 3 patients from the 5 that we thought were best for both MAP and ML categories. The 3 patients we will be choosing are patients 1, 4, and 5, as we feel they have the best data correlation and accuracies.

```
In [60]: def plot_joint_likelihoods(jointH0, jointH1, values1, values2, patient_num):
    n = len(values1)
    m = len(values2)

    x_vals = np.arange(n)
    y_vals = np.arange(m)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')

    fig = plt.figure(figsize=(14, 6))

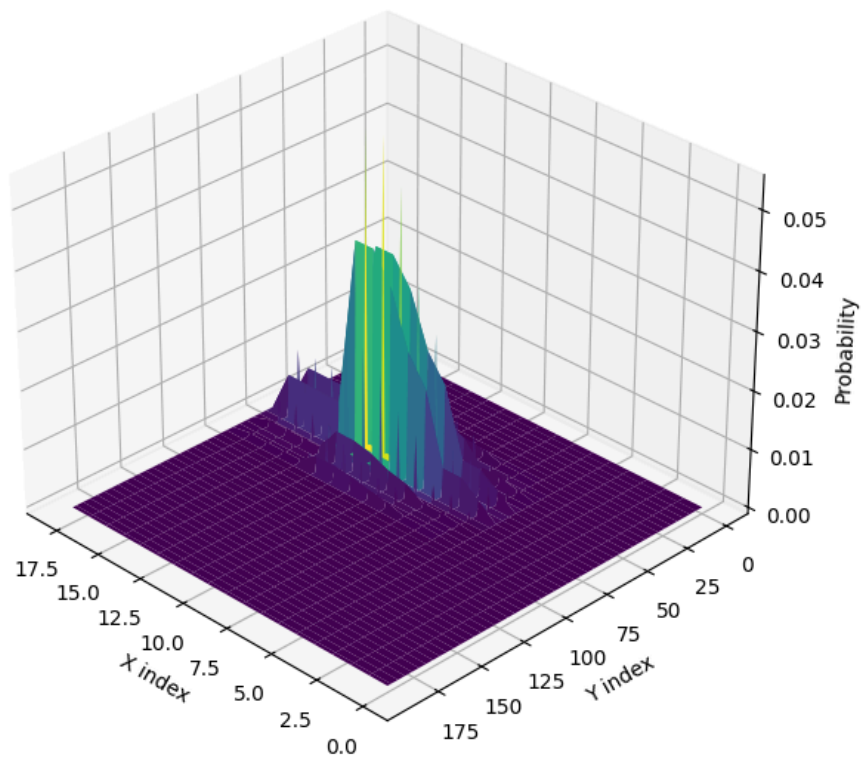
    ax1 = fig.add_subplot(1, 2, 1, projection='3d')
    ax1.plot_surface(X, Y, jointH1, cmap='viridis')
    ax1.set_title(f'Patient {patient_num} - P(X, Y | H1)')
    ax1.set_xlabel('X index')
    ax1.set_ylabel('Y index')
    ax1.set_zlabel('Probability')
    ax1.view_init(elev=30, azim=135)

    ax2 = fig.add_subplot(1, 2, 2, projection='3d')
    ax2.plot_surface(X, Y, jointH0, cmap='plasma')
    ax2.set_title(f'Patient {patient_num} - P(X, Y | H0)')
    ax2.set_xlabel('X index')
    ax2.set_ylabel('Y index')
    ax2.set_zlabel('Probability')
    ax2.view_init(elev=30, azim=135)

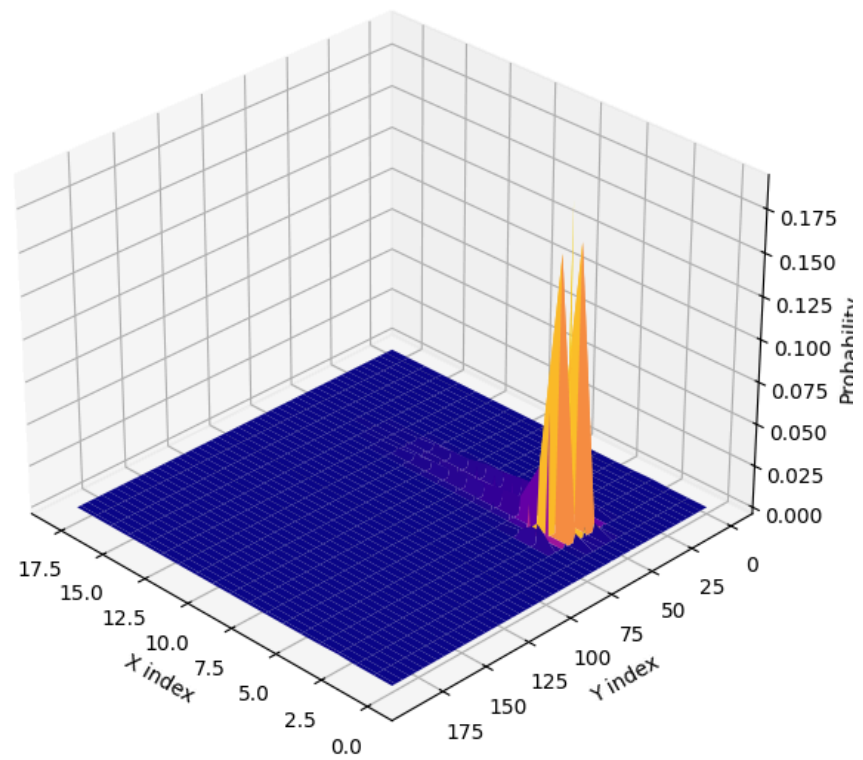
    plt.tight_layout()
    plt.show()
```

```
In [61]: plot_joint_likelihoods(pat1jointH0, pat1jointH1, pat1jointValues1, pat1jointValues2, patient_num=1)
plot_joint_likelihoods(pat4jointH0, pat4jointH1, pat4jointValues1, pat4jointValues2, patient_num=4)
plot_joint_likelihoods(pat5jointH0, pat5jointH1, pat5jointValues1, pat5jointValues2, patient_num=5)
```

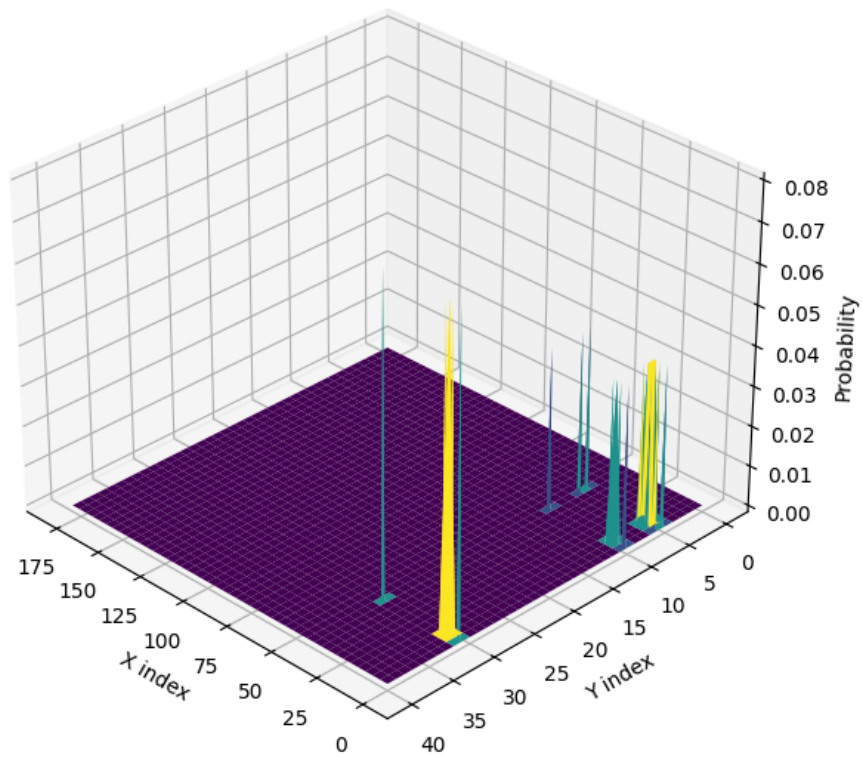
Patient 1 - $P(X, Y \mid H1)$



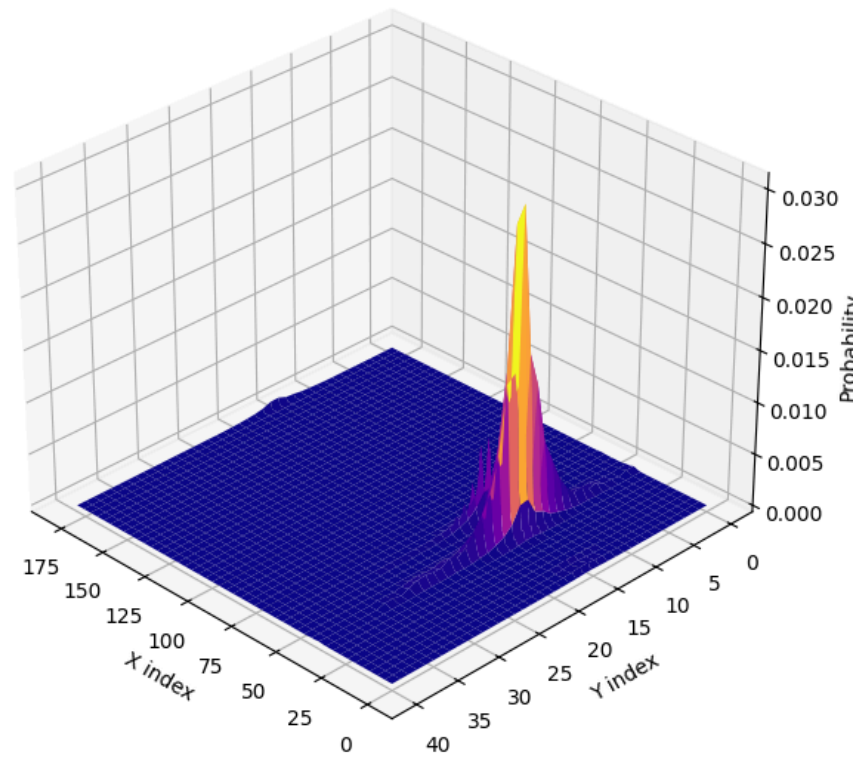
Patient 1 - $P(X, Y \mid H0)$

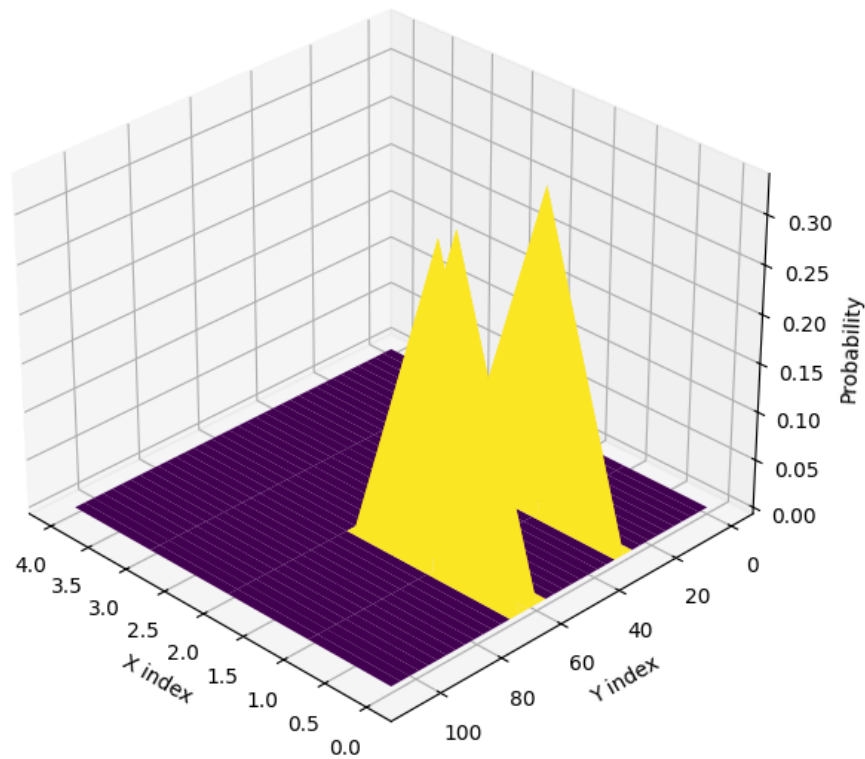
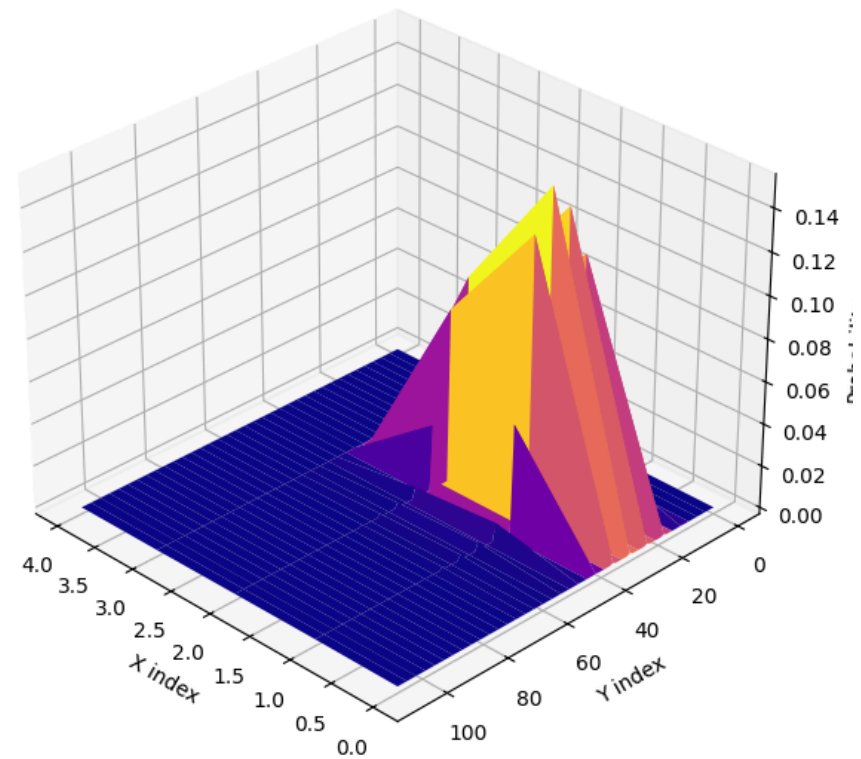


Patient 4 - $P(X, Y \mid H1)$



Patient 4 - $P(X, Y \mid H0)$



Patient 5 - $P(X, Y | H1)$ Patient 5 - $P(X, Y | H0)$ 

Task 4.2a - Joint Distribution Predictions

```
In [62]: def generateJointPreds(testData, f1Idx, f2Idx, jointHTTable):
    mlPreds = []
    mapPreds = []

    feature1_test = testData[f1Idx]
    feature2_test = testData[f2Idx]

    featurePairs = np.array(jointHTTable[:, [0, 1]])

    for val1, val2 in zip(feature1_test, feature2_test):
        matches = np.where((featurePairs[:, 0] == val1) & (featurePairs[:, 1] == val2))[0]
        if len(matches) > 0:
            idx = matches[0]
        else:
            dists = np.sqrt((featurePairs[:, 0] - val1)**2 + (featurePairs[:, 1] - val2)**2)
```

```

        idx = np.argmin(dists)

        mlPreds.append(int(jointHTTable[idx, 4]))
        mapPreds.append(int(jointHTTable[idx, 5]))

    return mlPreds, mapPreds

```

```

In [63]: pat1JointMLPreds, pat1JointMAPPreds = generateJointPreds(pat1TestData, 0, 2, pat1JointHT)
pat4JointMLPreds, pat4JointMAPPreds = generateJointPreds(pat4TestData, 1, 4, pat4JointHT)
pat5JointMLPreds, pat5JointMAPPreds = generateJointPreds(pat5TestData, 0, 2, pat5JointHT)

```

Task 4.2b & 4.3a - Error Probabilities & Plots

```

In [64]: def errorTableJoint(predsML, predsMAP, testLabels, priorH0, priorH1):
    falseAlarmsML = 0
    missDetectionsML = 0

    falseAlarmsMAP = 0
    missDetectionsMAP = 0

    for j in range(len(testLabels)):
        if predsML[j] == 0 and testLabels[j] == 1:
            missDetectionsML += 1
        if predsML[j] == 1 and testLabels[j] == 0:
            falseAlarmsML += 1

        if predsMAP[j] == 0 and testLabels[j] == 1:
            missDetectionsMAP += 1
        if predsMAP[j] == 1 and testLabels[j] == 0:
            falseAlarmsMAP += 1

    falseAlarmsML /= len(testLabels)
    missDetectionsML /= len(testLabels)
    falseAlarmsMAP /= len(testLabels)
    missDetectionsMAP /= len(testLabels)

    errorML = 0.5 * (falseAlarmsML + missDetectionsML)
    errorMAP = priorH0 * falseAlarmsMAP + priorH1 * missDetectionsMAP

    table = np.array([
        [falseAlarmsML, missDetectionsML, errorML],
        [falseAlarmsMAP, missDetectionsMAP, errorMAP]
    ])

```



```
return table
```

```
In [65]: pat1JointErrorTable = errorTableJoint(pat1JointMLPreds, pat1JointMAPPreds, pat1TestLabels, pat1PriorH0, pat1PriorH1)
pat4JointErrorTable = errorTableJoint(pat4JointMLPreds, pat4JointMAPPreds, pat4TestLabels, pat4PriorH0, pat4PriorH1)
pat5JointErrorTable = errorTableJoint(pat5JointMLPreds, pat5JointMAPPreds, pat5TestLabels, pat5PriorH0, pat5PriorH1)
```

```
In [66]: import matplotlib.pyplot as plt

def plotAlarms(predsML, predsMAP, testLabels, patient_num):
    fig, axes = plt.subplots(3, 1, figsize=(12, 8), sharex=True)

    axes[0].bar(range(len(predsML)), predsML, color='blue')
    axes[0].set_title(f'Patient {patient_num} - ML Rule Alarms')

    axes[1].bar(range(len(predsMAP)), predsMAP, color='green')
    axes[1].set_title(f'Patient {patient_num} - MAP Rule Alarms')

    axes[2].bar(range(len(testLabels)), testLabels, color='red')
    axes[2].set_title(f'Patient {patient_num} - Golden Alarms (True Labels)')

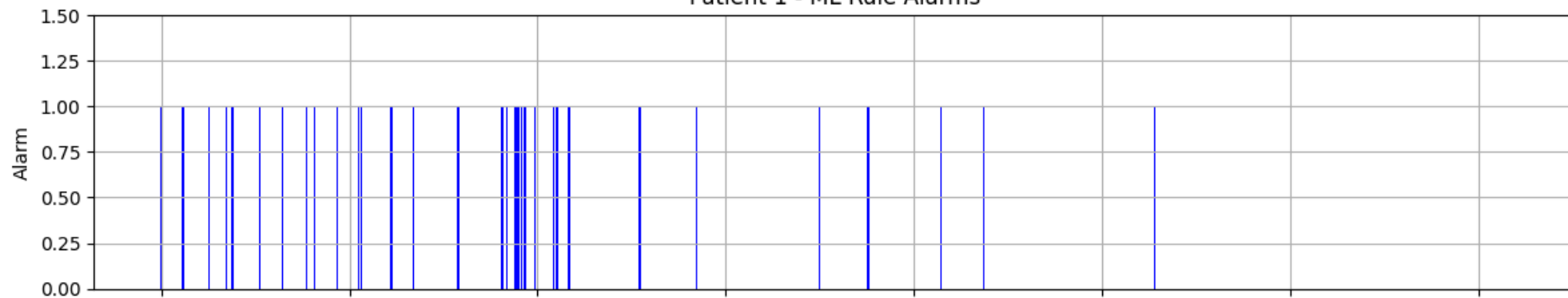
    for ax in axes:
        ax.set_ylim(0, 1.5)
        ax.set_ylabel('Alarm')
        ax.grid(True)

    axes[2].set_xlabel('Test Sample Index')

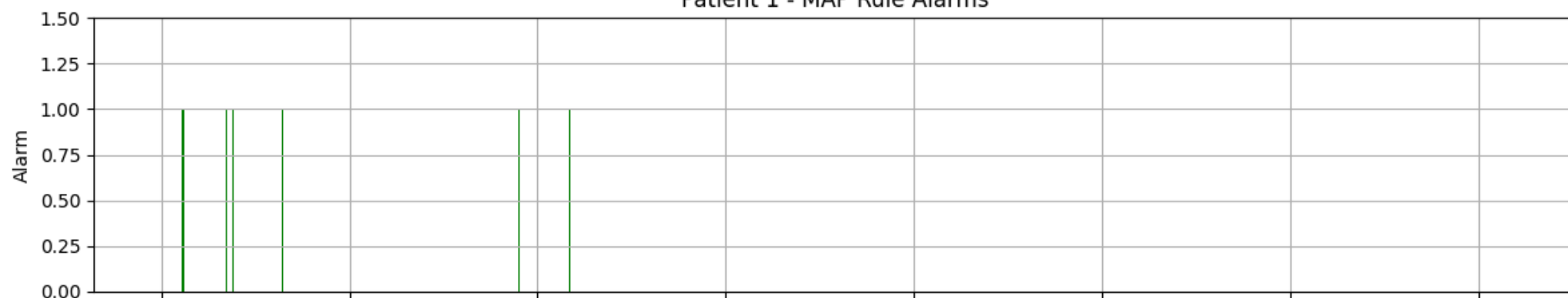
    plt.tight_layout()
    plt.show()
```

```
In [67]: plotAlarms(pat1JointMLPreds, pat1JointMAPPreds, pat1TestLabels, patient_num=1)
plotAlarms(pat4JointMLPreds, pat4JointMAPPreds, pat4TestLabels, patient_num=4)
plotAlarms(pat5JointMLPreds, pat5JointMAPPreds, pat5TestLabels, patient_num=5)
```

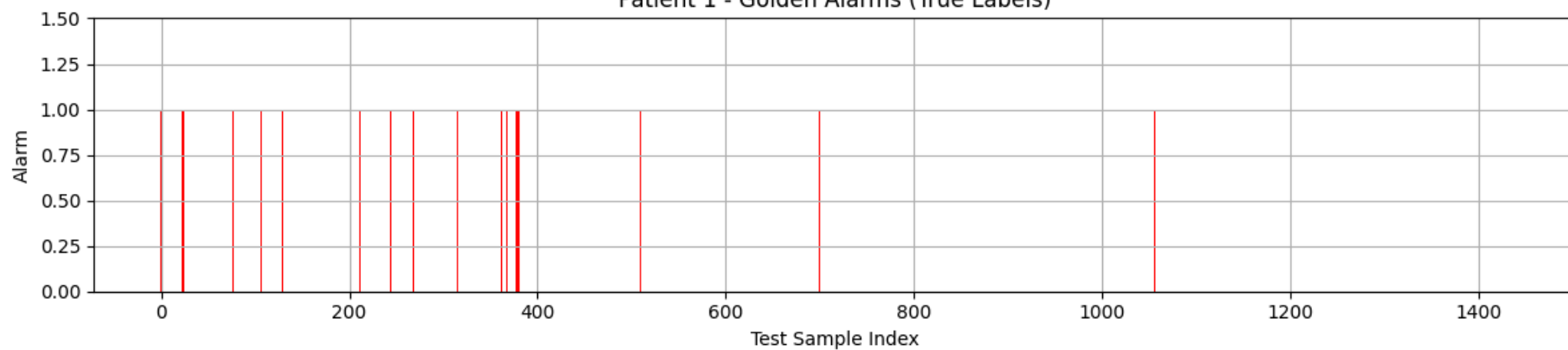
Patient 1 - ML Rule Alarms



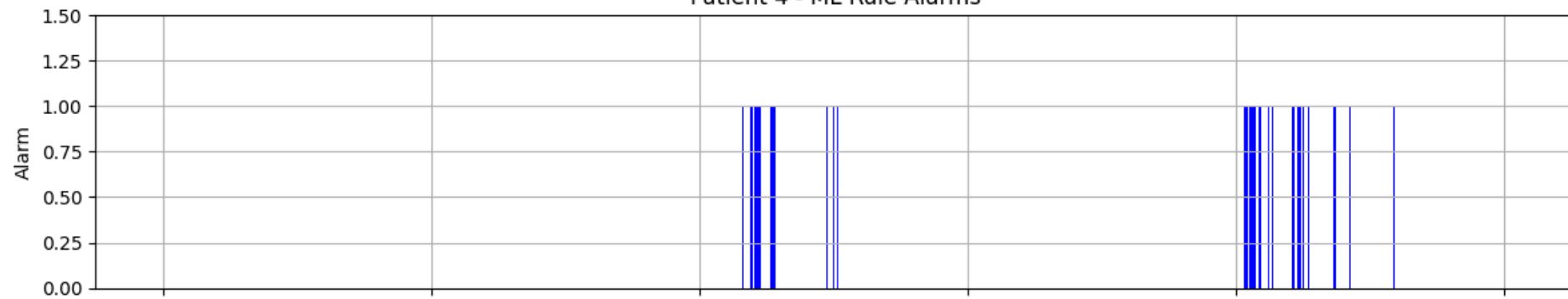
Patient 1 - MAP Rule Alarms



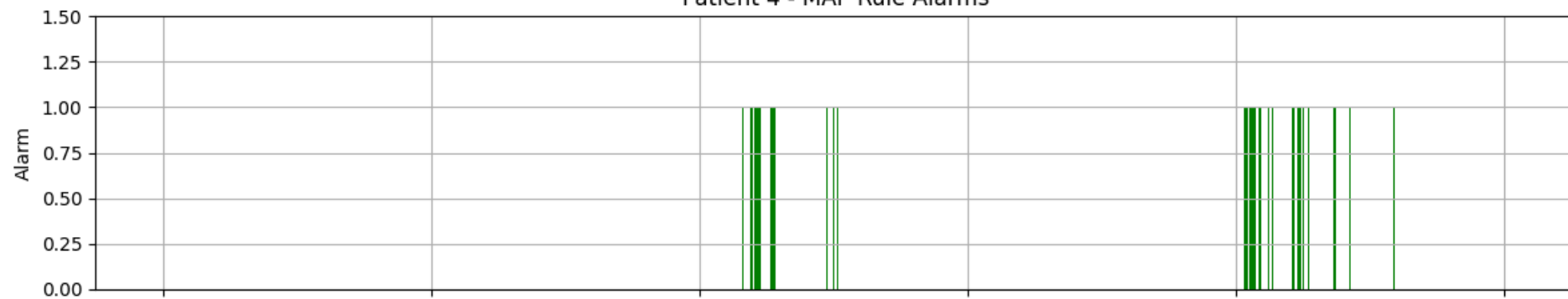
Patient 1 - Golden Alarms (True Labels)



Patient 4 - ML Rule Alarms

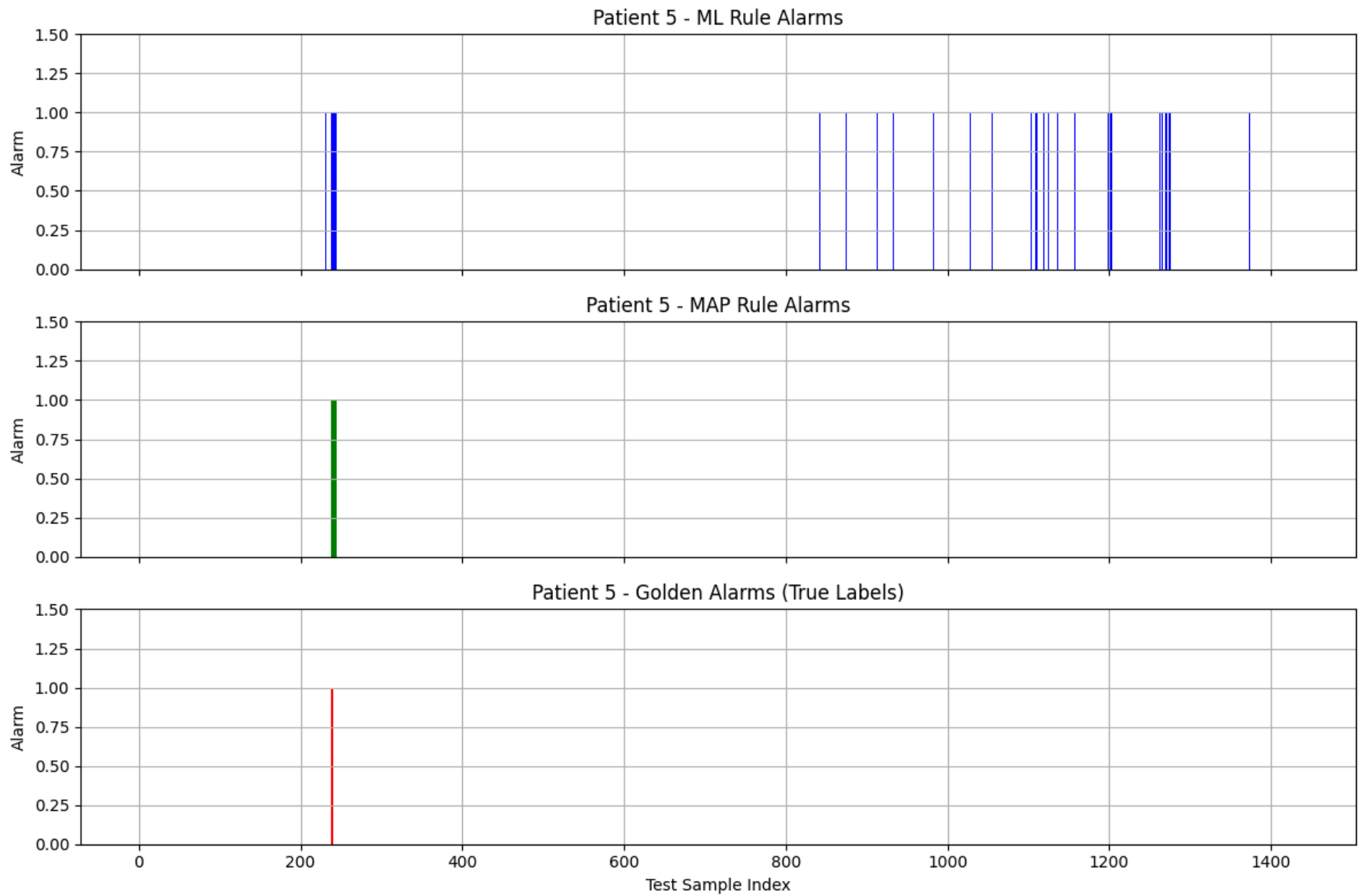


Patient 4 - MAP Rule Alarms



Patient 4 - Golden Alarms (True Labels)





```
In [72]: print("Patient 1 Error Table:")
print(pat1JointErrorTable)
print("=====")

print("Patient 4 Error Table:")
print(pat4JointErrorTable)
print("=====")
```

```
print("Patient 5 Error Table:")
print(pat5JointErrorTable)
```

Patient 1 Error Table:

```
[[0.03977669 0.00139567 0.02058618]
 [0.00418702 0.0132589  0.00445918]]
```

=====

Patient 4 Error Table:

```
[[0.03784861 0.          0.0189243 ]
 [0.03784861 0.          0.03784861]]
```

=====

Patient 5 Error Table:

```
[[0.03277545 0.00069735 0.0167364 ]
 [0.00348675 0.00069735 0.00348675]]
```

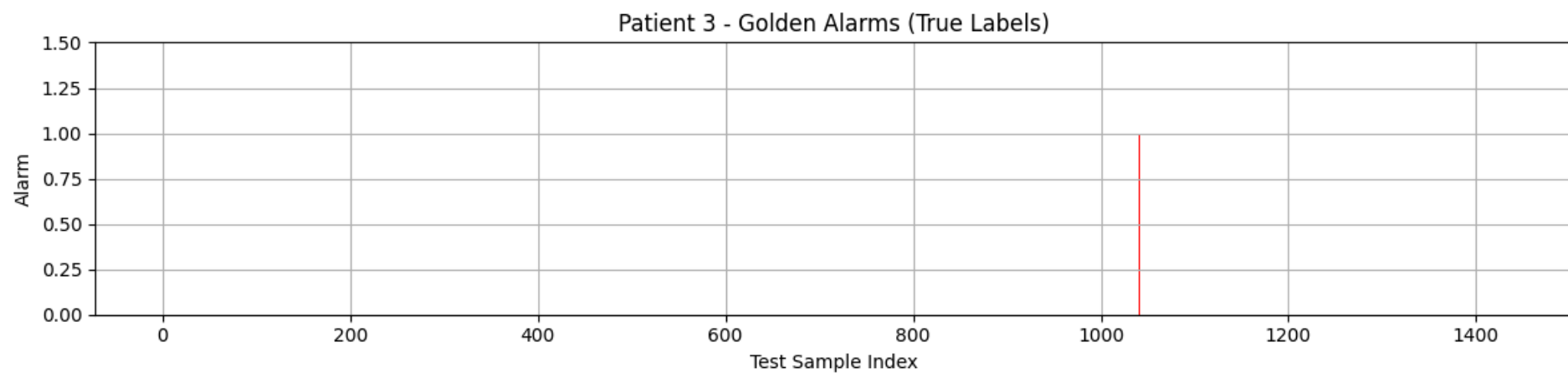
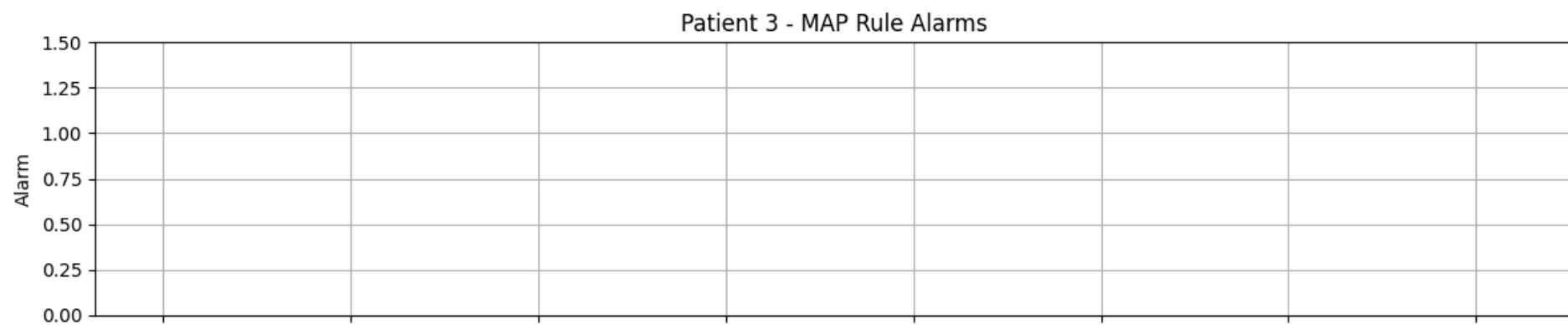
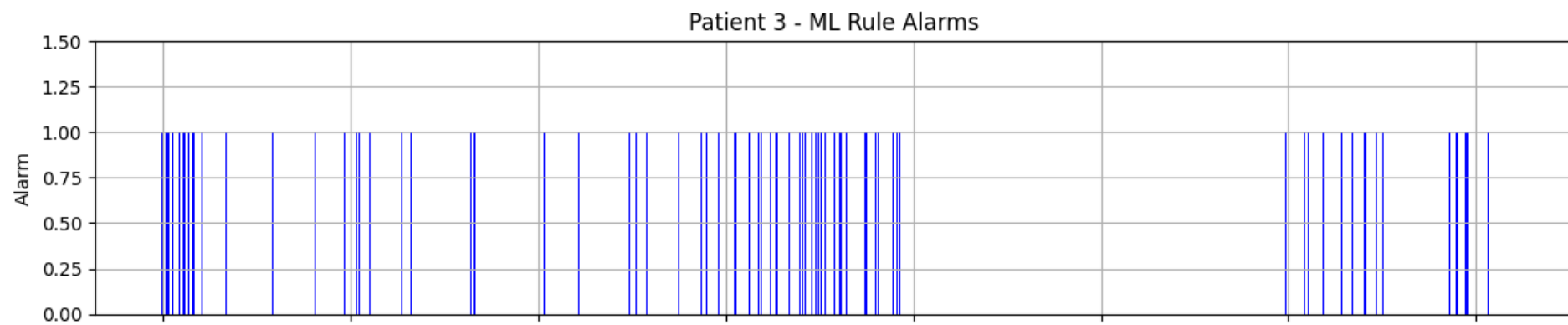
Task 4.3b - Criteria Analysis

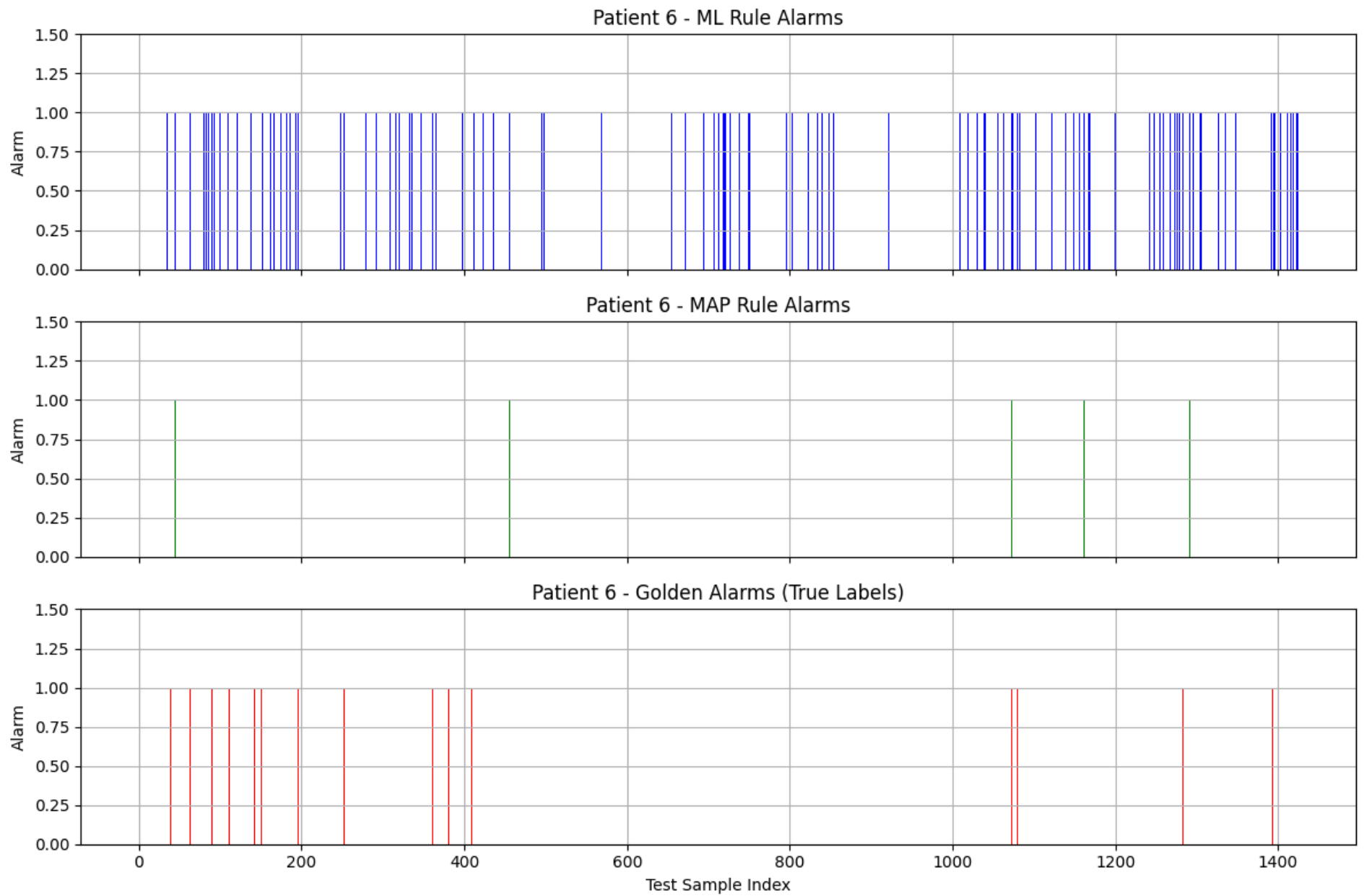
Let us first look at the plots and error probabilities of the 2 patients that we left out: patients 3 and 6.

```
In [76]: pat3JointMLPreds, pat3JointMAPPreds = generateJointPreds(pat3TestData, 0, 6, pat3JointHT)
pat6JointMLPreds, pat6JointMAPPreds = generateJointPreds(pat6TestData, 3, 5, pat6JointHT)

pat3JointErrorTable = errorTableJoint(pat3JointMLPreds, pat3JointMAPPreds, pat3TestLabels, pat3PriorH0, pat3PriorH1)
pat6JointErrorTable = errorTableJoint(pat6JointMLPreds, pat6JointMAPPreds, pat6TestLabels, pat6PriorH0, pat6PriorH1)

plotAlarms(pat3JointMLPreds, pat3JointMAPPreds, pat3TestLabels, patient_num=3)
plotAlarms(pat6JointMLPreds, pat6JointMAPPreds, pat6TestLabels, patient_num=6)
```





```
In [77]: print("Patient 3 Error Table:")
print(pat3JointErrorTable)
print("=====")

print("Patient 6 Error Table:")
print(pat6JointErrorTable)
```

Patient 3 Error Table:

```
[[0.09636872 0.00139665 0.04888268]
 [0.          0.00139665 0.          ]]
```

=====

Patient 6 Error Table:

```
[[0.13403509 0.00701754 0.07052632]
 [0.0077193  0.01263158 0.00781754]]
```

As can be seen, these error probabilities are much higher than the error probabilities of the patients that we selected. So, the patients that we have selected and their features are a good choice. Anything below 5% accuracy is generally a good threshold to meet, and all errors of the patients that we have selected meet this criteria. Therefore, there is no reason to redetermine features or retrain the prediction process.

Task 4.3c - Average Error Probabilities

```
In [80]: patient1_error_table = pat1JointErrorTable
patient4_error_table = pat4JointErrorTable
patient5_error_table = pat5JointErrorTable

all_error_tables = [patient1_error_table, patient4_error_table, patient5_error_table]

ml_errors = [table[0, 2] for table in all_error_tables]
map_errors = [table[1, 2] for table in all_error_tables]

average_ml_error = np.mean(ml_errors)
average_map_error = np.mean(map_errors)

print(f"Average ML Error across 3 patients: {average_ml_error:.6f}")
print(f"Average MAP Error across 3 patients: {average_map_error:.6f}")
```

Average ML Error across 3 patients: 0.018749

Average MAP Error across 3 patients: 0.015265

Task 4.3d - Insights

In this project, we analyzed the detection performance for Patients 1, 4, and 5 based on selected feature pairs. The selection of feature pairs was guided both by low error rates from Task 3.2 and by the feature-golden alarm correlation analysis from Task 3. For Patient 1, features 0 and 2 were chosen due to their strong correlation with the labels and relatively low likelihood error. Similarly, for Patient 4, features 1 and 4 were selected, and for Patient 5, features 0 and 2 were used to minimize error and joint probability error.

Using these selected pairs, we generated alarms based on both ML and MAP decision rules, evaluated their performance, and computed false alarm, miss detection, and total error rates. The error tables showed that for Patient 1, the ML probability of error was approximately

2.06% and the MAP probability of error was approximately 0.45%. For Patient 4, the ML probability of error was approximately 1.89%, while the MAP probability of error was approximately 3.78%. For Patient 5, the ML probability of error was approximately 1.67%, and the MAP probability of error was approximately 0.35%. The average probability of error across all three patients was 1.87% for ML and 1.53% for MAP, confirming that MAP generally outperformed ML except for Patient 4.

A closer analysis reveals that the MAP rule was particularly effective in reducing both false alarms and miss detections when feature priors were correctly aligned with feature behavior, as seen in Patients 1 and 5. However, in Patient 4, MAP performance worsened compared to ML, most likely due to prior values over-weighting the likelihood of one class over the other, leading to a higher false alarm rate. This observation shows that while MAP typically provides better performance by incorporating prior probabilities, it is sensitive to poorly assigned prior probabilities. Overall, carefully selecting feature pairs based on both prediction separation and correlation with the actual answer significantly improved the detection system's performance.