Matthew Euliano 5/31/2022

EECE5644 – Intro to Machine Learning – Homework 1

Contents

Matthew Euliano 5/31/2022

EECE5644 – Intro to Machine Learning – Homework 1

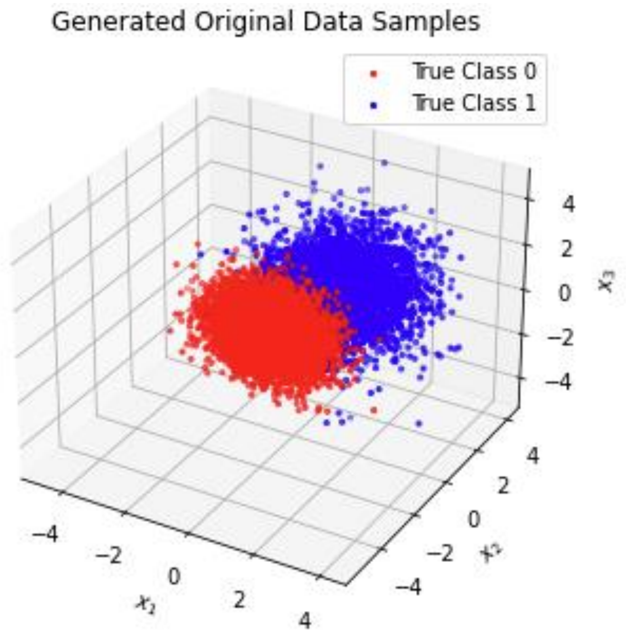
Problem 1

For this problem, a 10,000-sample dataset was generated accorded to a multivariate gaussian distribution with the pre-defined priors, means, and covariance matrices below:

$$\mu_0 = \begin{bmatrix} -1/2 \\ -1/2 \\ -1/2 \end{bmatrix}, \Sigma_0 = \begin{bmatrix} 1 & -0.5 & 0.3 \\ -0.5 & 1 & -0.5 \\ 0.3 & -0.5 & 1 \end{bmatrix},\ \mu_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0.3 & -0.2 \\ 0.3 & 1 & 0.3 \\ -0.2 & 0.3 & 1 \end{bmatrix}, \begin{matrix} P(L=0) = 0.65 \\ P(L=1) = 0.35 \end{matrix}$$

A 3-dimensional random vector **X** was created according to: *p(x)=p(**x**|L=0)P(L=0)+p(**x**|L=1)p(L=1)*, where L is the true class label of the pdf that generated the data. The class-conditional PDFs are gaussian and defined as $\mathrm{p}(\mathbf{x}|\mathrm{L}=0) = \mathrm{g}(\mathbf{x}|\mu_0, \Sigma_0)$ and $\mathrm{p}(\mathbf{x}|\mathrm{L}=1) = \mathrm{g}(\mathbf{x}|\mu_1, \Sigma_1)$

The generated dataset is shown below:



With Label 0 = 6484 samples, Label 1 = 3516 samples

Part A: ERM classification using knowledge of true data PDF

The minimum expected risk classification rule in the form of a likelihood ratio test:

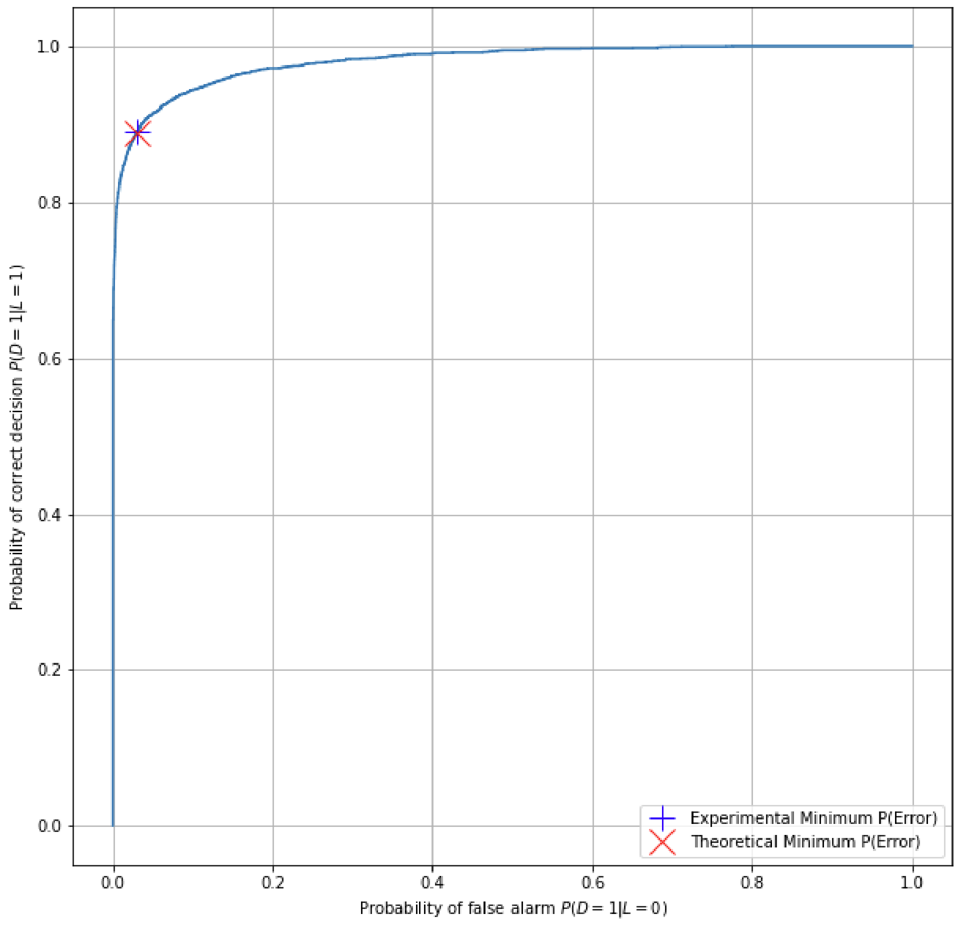
$$\frac{p(x|L=1)}{p(x|L=0)} \begin{matrix} > \\ < \end{matrix} \frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}} \frac{P(L=0)}{P(L=1)} = \frac{1-0}{1-0} * \frac{0.65}{0.35} = 1.86 = \gamma$$

In this case, the cost for incorrect classifications were set to 1.

The classifier was implemented on the generated data and the γ-threshold was varied using the sorted discriminant score, which represents all the possible values of gamma from each point in the dataset.

Iterating through, decisions were made for each observation for each γ, and the decisions were compared to the true labels to find the number of true positives ($P(D=1|L=1;\gamma)$ and false positives $P(D=1|L=0;\gamma)$.

Using these paired values, an approximation of the ROC curve of the classifier was created:



The locations of the theoretical minimum error in addition to the minimum error determined by the parametric sweep of gamma are plotted on the ROC curve.

The minimum probability of error was calculated using:

$$P(error;\gamma) = P(D=1|L-0;\gamma)P(L=1) + P(D=0|L=0)P(L=1)$$

The resulting values are:

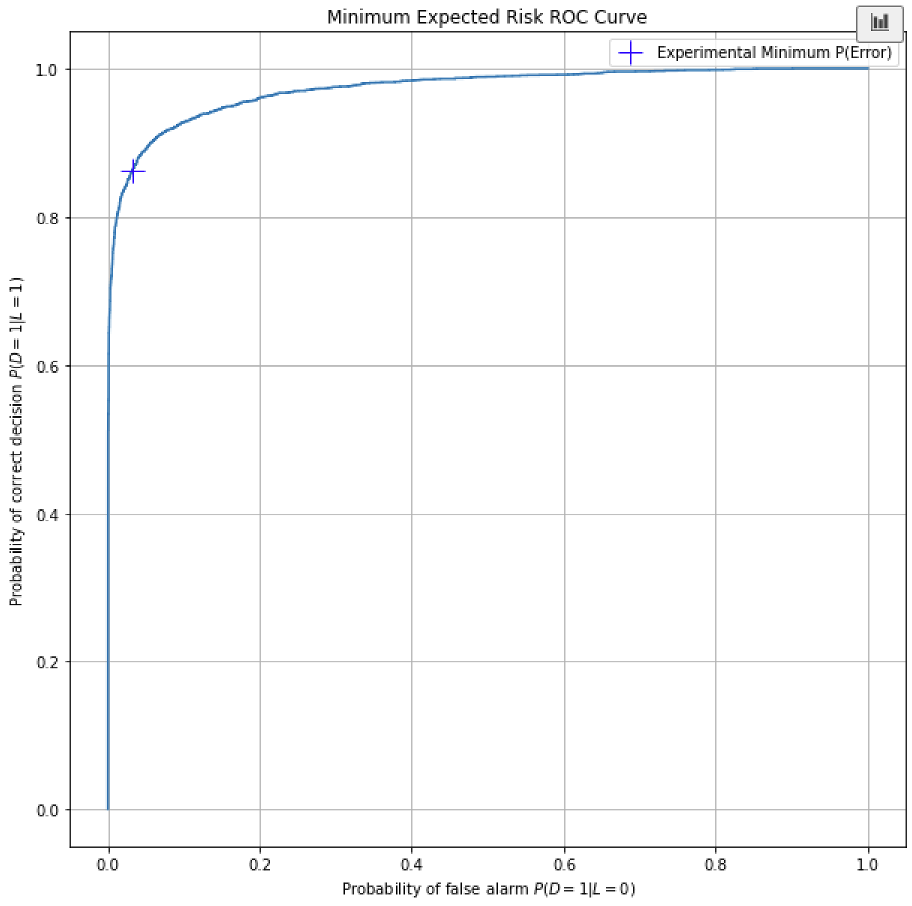
	γ	$\min P(error;\gamma)$
Theoretical	1.857	0.0583
Full Knowledge Estimate	1.845	0.0579

These values are very close, as expected, because the ERM classifier has full knowledge of the true data, including mean, variance, and priors. The theoretical optimal is seen to have a higher minimum probability of error, which may be surprising. This is likely due to the theoretical optimal values being generated according to a dataset that perfectly matches the priors (e.g. a 6500-sample and 3500-sample split). Because the theoretical optimal is being projected onto the ROC curve with a randomly generated dataset, it might not actually appear optimal. As the number of samples is increased, it would be expected for the theoretical optimal values to be more optimal than that estimated from data.

Part B: ERM classification using incorrect knowledge – Naïve Bayesian Classifier

Here, we assume we know the true class prior probabilities and the means of the gaussian distributions, but we incorrectly assume the covariance matrices are both equal to the diagonal, which in this case is just the identity matrix.

The steps in part A were repeated using the same data, but now analyzed using a Naïve Bayesian approach. The ROC curve with the approximated covariance matrices is plotted below:



The location of the minimum error determined by a parametric sweep of gamma for the Naïve Bayes case is marked on the plot.

	γ	$\min P(error;\gamma)$
Theoretical	1.857	0.0583
Full Knowledge Estimate	1.845	0.0579
Naïve Bayes Estimate	0.964	0.0678

The minimum achievable errors and γ value in this case begin to diverge compared to the previous example due to the misrepresentation of data by assuming identity matrices for the covariance matrices. However, the minimum percent error value remains reasonably close to the theoretical optimum because the overlap in the two datasets was not significantly altered when the covariances were changed. Therefore, the classifier is still able to correctly identify most cases.

Part C: Use Fisher Linear Discriminant Analysis (LDA) – based classifier

These steps were repeated, once again, but this time using the Fisher LDA. In this approach, the two datasets being classified are projected into a single dimension that maximizes the ratio of the difference in means to variances. As a result, the overlap between the datasets is minimized, making classification easier.

First, the class-conditional pdf means and covariance matrices were estimated using sample average estimators. These were then used to determine the Fisher LDA projection weight vector using eigen decomposition between class scatter matrices (w_{LDA}). For the classification rule $w_{LDA}^T x$ compared to a threshold τ that varies similarly to the first two parts to create another ROC curve.

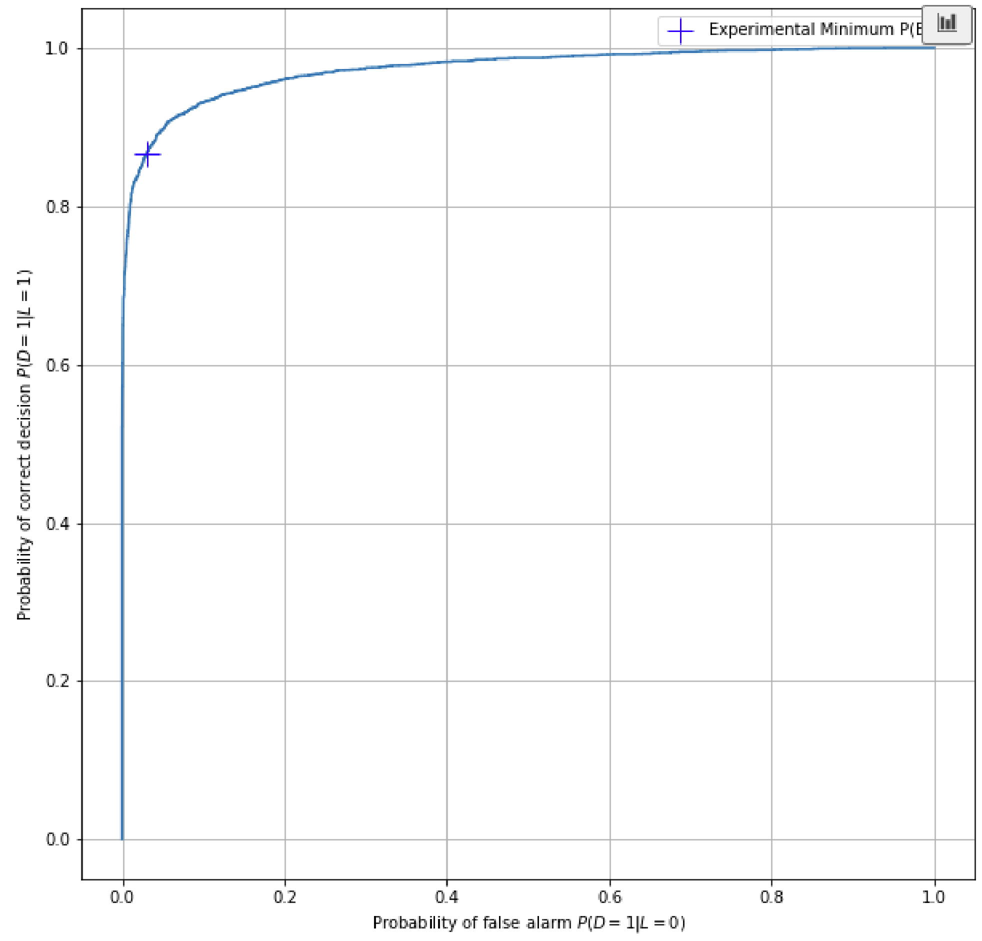
Fisher LDA Classification rule:

$$\boldsymbol{w}_{LDA}\boldsymbol{x} \underset{<}{\overset{>}{}} \tau$$

Where w_{LDA} is the generalized eigenvector of the scatter matrices S_B *and* S_w, with the largest eigenvalue as described by:

$$S_w^{-1} S_B W = \lambda W \quad , \quad S_w = \Sigma_0 + \Sigma_1 \quad , \quad S_B = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T$$

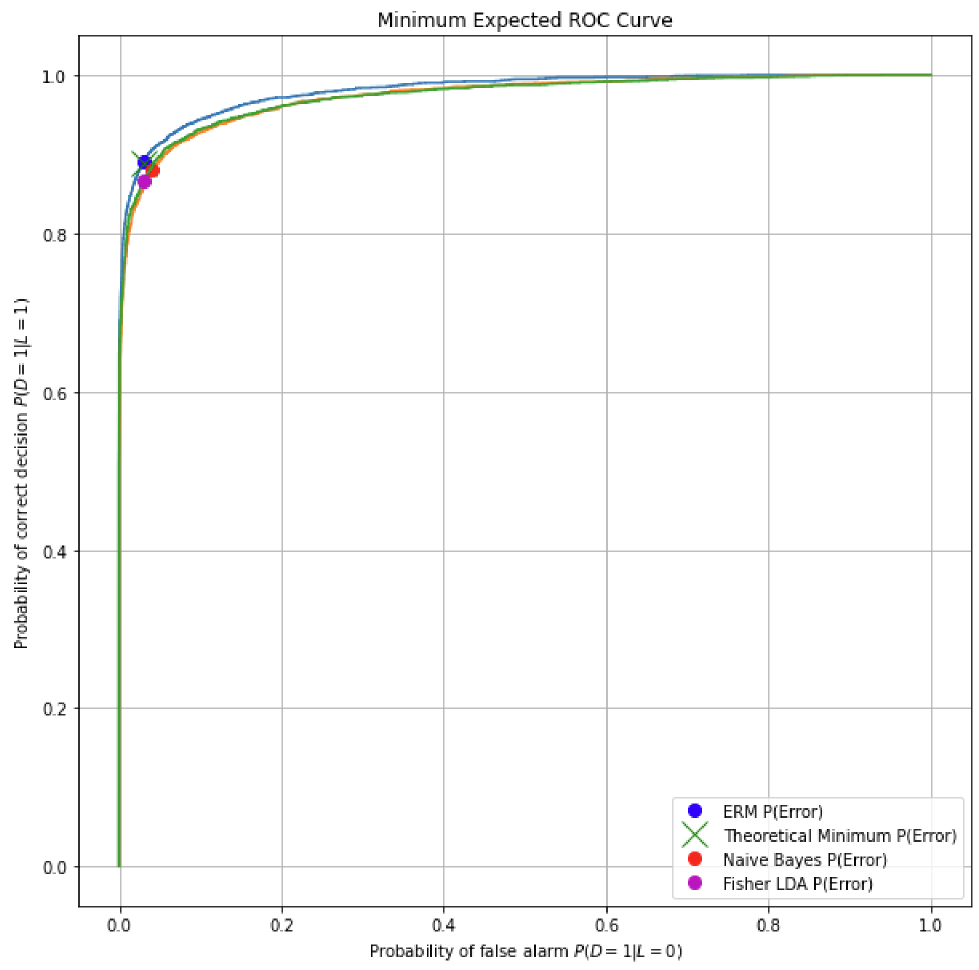
The resulting ROC curve can be seen below:



The location of the minimum error determined by a parametric sweep of gamma for the dimensionality-reduced Fisher LDA case is marked on the plot.

	γ	$\min P(error;\gamma)$
Theoretical	1.857	0.0583
Full Knowledge Estimate	1.845	0.0579
Naïve Bayes Estimate	0.964	0.0678
Fisher LDA Estimate	1.599	0.0658

Comparing the three methods (ERM with known covariance, ERM with Naïve Bayes, and Fisher LDA), it was determined that the Fisher LDA estimate is an improvement upon the Naïve Bayes estimate while being less optimal than the full-knowledge ERM estimate. To further showcase these results, the ROC curves were plotted together. The ideal case is at the top left corner, so the results above can be seen visually to follow the trends described previously.



Problem 2

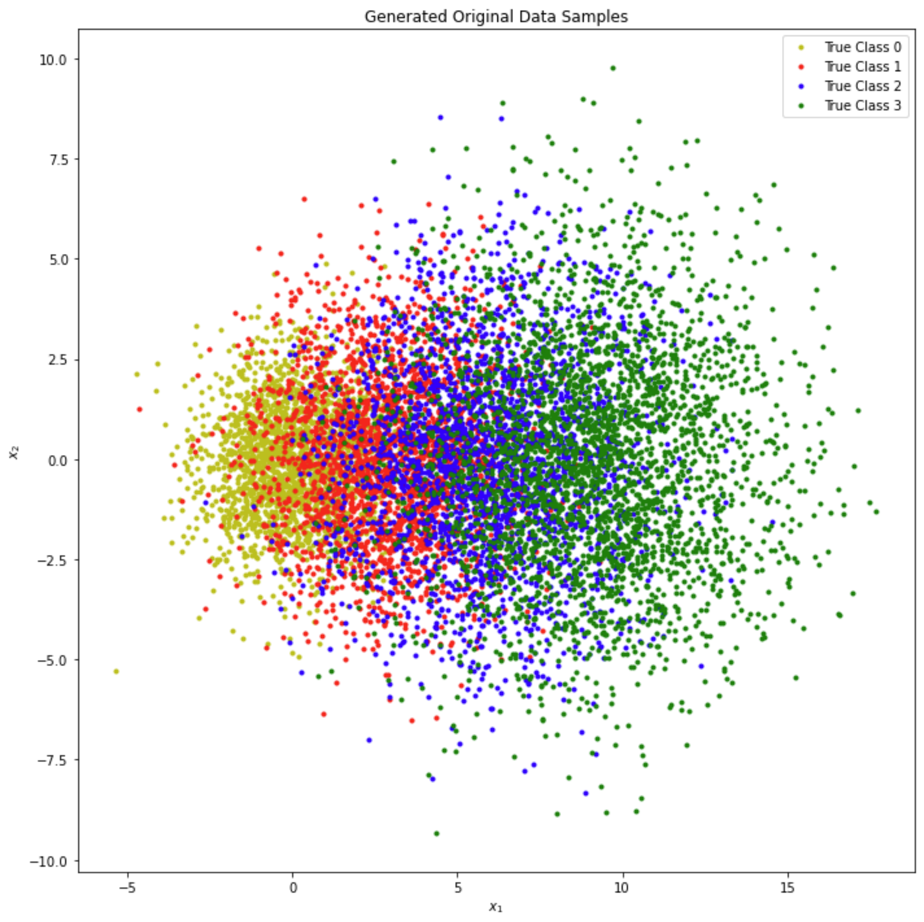
A 2-dimensional random vector **X** takes values from a mixture of four gaussians with provided priors. The mean and covariance matrices of those four gaussians were arbitrarily defined based on the mean vectors being approximately equally spaced along a line and the covariance matrices being scaled versions of the identity matrix, allowing for sufficient overlap between distributions.

They were selected as follows:

$$\mu = \begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}3\\0\end{bmatrix}, \begin{bmatrix}6\\0\end{bmatrix}, \begin{bmatrix}9\\0\end{bmatrix} \qquad \Sigma = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}, \begin{bmatrix}4 & 0\\0 & 4\end{bmatrix}, \begin{bmatrix}6 & 0\\0 & 6\end{bmatrix}, \begin{bmatrix}8 & 0\\0 & 8\end{bmatrix}$$

With priors: *P(L) = 0.2, 0.25, 0.25, 0.3*

10,000 samples were generated tracking true labels, which can be seen below:



The number of samples per label: Label 0 = 1974, Label 1 = 2427, Label 2 = 2500, Label 3 = 3099

Part A: Minimum Probability of Error Classification

The samples were analyzed using the ERM decision rule that achieves minimum probability of error (0-1 loss), also known as the Bayes Decision Rule or MAP Classifier, which is defined as:

$$D(x) = \underset{D \in \{1,2,3,4\}}{argmin} \sum_{i=1}^{C} \lambda_{Dl}\, p(L = l|x)$$

Where C is the number of classes, λ_{DL} is the loss for classifying a point from label l in class D, and P(L=l|x) is the class posteriors which are calculated using:

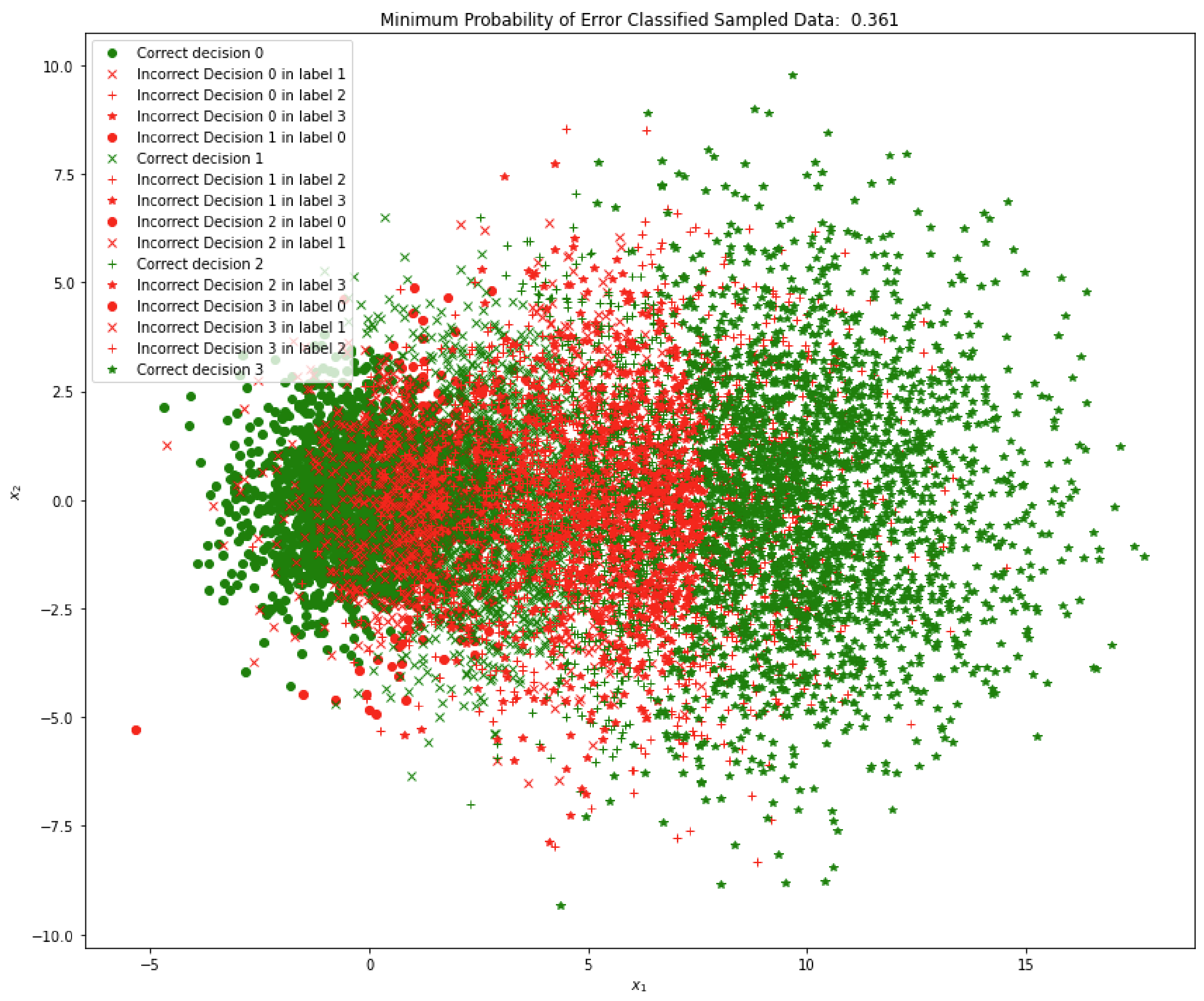
$$p(L = l|x) = \frac{p(x|L = l)p(L = l)}{p(x)}$$

Where p(x|L=l) is the class-conditional, P(L=1) is the class prior, and $p(x) = \sum_{j=1}^{C} p(x|L = j)p(L]j)$.

The cost function used in the classification was chosen to minimize the probability of error with the loss matrix consisting of values of 0 or 1, such that any misclassification is equally weighted:

$$\Lambda = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

The data and the resulting classification decisions are shown below.



The classification visualization above indicates the four classes and which points are correctly and incorrectly categorized.

Next, running our data through the decision rule, we get a confusion matrix using ratios that show the probability of classification of each class:

		Truth			
		1	**2**	**3**	**4**
Decision	**1**	0.828	0.207	0.026	0.002
	2	0.171	0.588	0.260	0.055
	3	0.001	0.190	0.420	0.207
	4	0.000	0.015	0.294	0.736

With the minimum probability of error, or the expected Risk was calculated to be 0.3607 using the formula below:

$$E(Loss) = \frac{1}{N}\sum_{j=1}^{k}\sum_{i=1}^{n}\lambda_{ij}Num\_Decision_{ij}$$

Where N is the number of data points, λ_{ij} is the cost associated with a particular I decision for a j Label, and NumDecisions is the number of decisions for a particular combination of classification and label.

Part B: Higher Cost for Difference Between Means Classification

The same data was used as in part A using a cost function which increased the cost for misclassification between gaussians whose means have higher separation. The loss matrix used was changed to be:

$$\Lambda = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

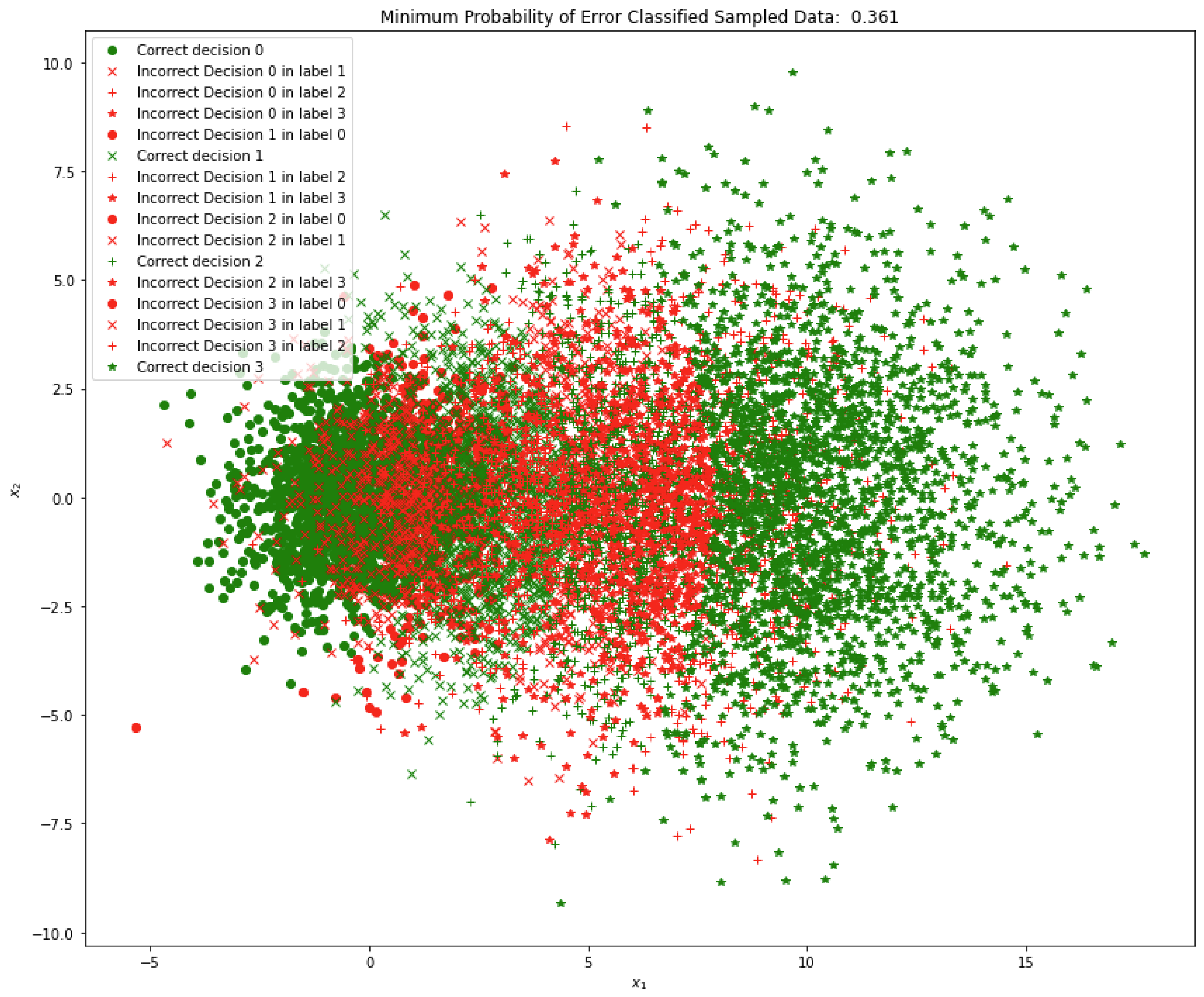
This increases the cost associated with misclassifying between distributions that are further away, such as deciding on label 1 when it is actually label 4 – this is defined to have the greatest cost of 3.

The resulting confusion matrix shows the probability of classification for each class.

		Truth			
		1	**2**	**3**	**4**
Decision	**1**	0.796	0.183	0.021	0.002
	2	0.202	0.577	0.210	0.043
	3	0.002	0.248	0.506	0.245
	4	0.000	0.011	0.263	0.710

The expected risk was calculated in the same way as part A, which was found to be 0.3611.

As expected, not much changes between the confusion matrices in parts A and B. This is because most of the error was between data distributions that were adjacent. With the new lambda matrix, adjacent data distributions still have a cost value of 1. For the few cases in which a misclassification was made beyond an adjacent distribution, the cost was increased, which can be seen in a slight increase in expected risk in the latter case. The visualization below is virtually the same as in Part A for this reason.


Minimum Probability of Error Classified Sampled Data: 0.361
Correct decision 0
Incorrect Decision 0 in label 1
Incorrect Decision 0 in label 2
Incorrect Decision 0 in label 3
Incorrect Decision 1 in label 0
Correct decision 1
Incorrect Decision 1 in label 2
Incorrect Decision 1 in label 3
Incorrect Decision 2 in label 0
Incorrect Decision 2 in label 1
Correct decision 2
Incorrect Decision 2 in label 3
Incorrect Decision 3 in label 0
Incorrect Decision 3 in label 1
Incorrect Decision 3 in label 2
Correct decision 3
x_1
x_2

Problem 3

This problem used the Wine Quality dataset and the Human Activity Recognition datasets. A minimum-probability-of-error classifier was used, assuming gaussian class conditional pdf of features for each class.

First, the sample-based estimates of priors, mean vectors, and covariance matrices were derived from the provided datasets to estimate the true mean and variance using the equations below:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \qquad s^2 = \hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$$

To compensate for ill-conditioned covariance matrices, a regularization term was added to the covariance estimate: $C_{Reg} = C_{Avg} + \lambda I$, where λ is a small positive regularization parameter that ensures the covariance matrix has all eigenvalues larger than this parameter.

With these trained Gaussian class-conditional pdfs and priors, the minimum-P(error) classification rule was applied on all training samples, from which we can learn the total number of errors, an estimate for error probability, a confusion matrix, and a visualization of the data using a subset of the features.

Next, dimensionality reduction was performed using principal component analysis (PCA) and these steps were repeated. To perform PCA, the data was first normalized to have a mean of 0 and a standard deviation of one. The orthogonal principal eigenvectors of the Σ-matrix were computed, and the data was projected onto each component. The RMSE, sum of eigenvalues, and fraction of variance was plotted against the number of principal components used.

The results and discussion for each dataset can be found below with some conclusions in the last section.

White Wine Dataset

The White Wine dataset has a total of 4898 samples following the distribution below. From that, the priors were calculated. It can be seen that labels 1, 2, and 10 were never labeled, so there will never be any decisions made for those labels. Therefore, in the code attached to this report, all analysis was done using labels 3 through 7, but for the sake of completeness, the confusion matrix has been expanded to include those empty labels.

Label	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Num Samples	0	0	20	163	1457	2198	880	175	5	0
Priors	0	0	0.0041	0.0333	0.2975	0.4488	0.1797	0.0357	0.0010	0

The total number of misclassifications was found to be 2315, which corresponds to an error estimate of 0.4726. This means that almost *half* of the samples were misclassified!

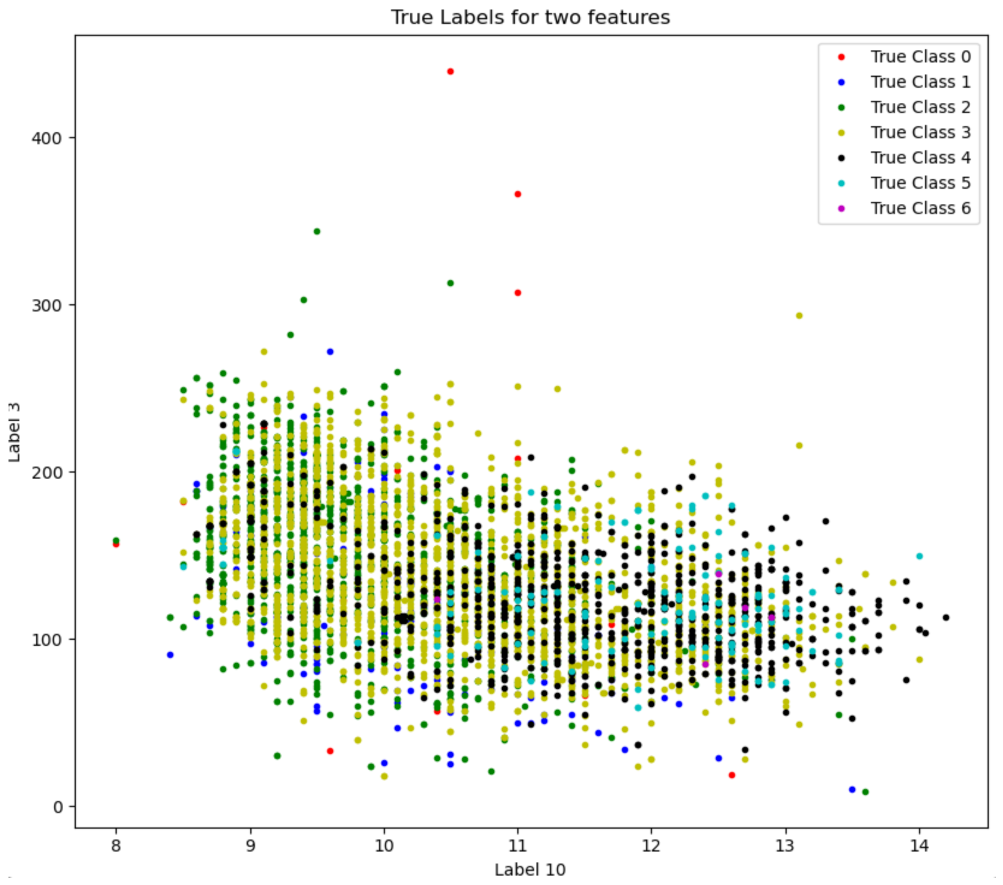
Matthew Euliano 5/31/2022

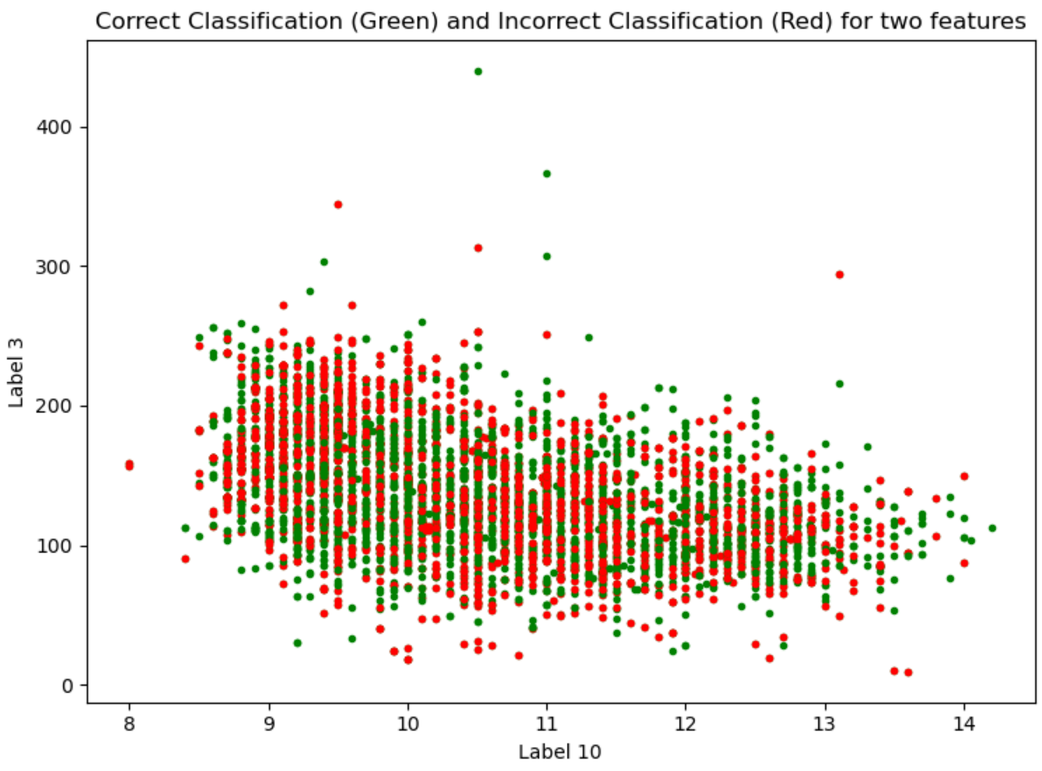
EECE5644 – Intro to Machine Learning – Homework 1

The Confusion Matrix below shows the probabilities of correctly or incorrectly classifying samples.

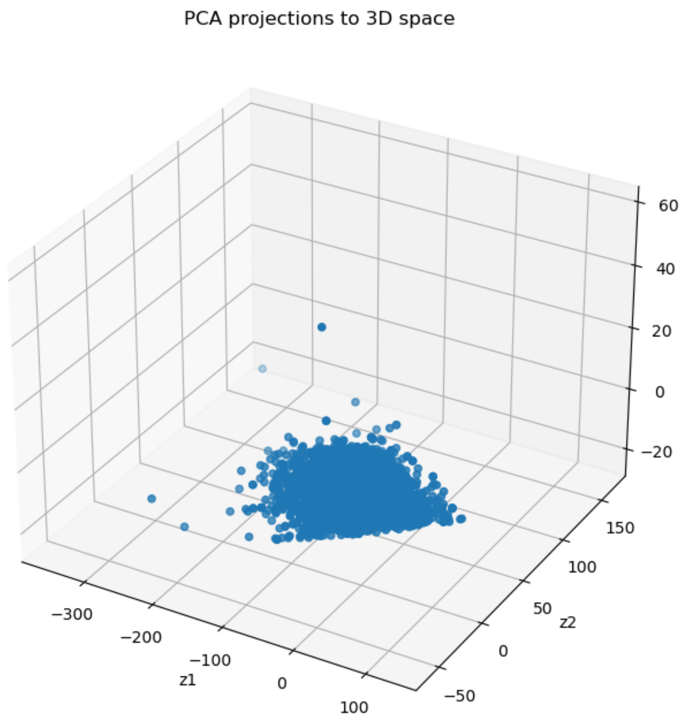
		Truth									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Decision	**1**	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0
	3	0	0	0.350	0.012	0.003	0.005	0.002	0.023	0	0
	4	0	0	0.050	0.018	0.005	0.003	0	0	0	0
	5	0	0	0.250	0.650	0.638	0.280	0.116	0.091	0	0
	6	0	0	0.350	0.307	0.34	0.609	0.535	0.451	0.400	0
	7	0	0	0	0.012	0.013	0.103	0.343	0.417	0.600	0
	8	0	0	0	0	0.001	0	0.003	0.017	0	0
	9	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0

The plots below show a visualization of the white wine dataset using a subset of features, which in this case is Label 3 ("Residual Sugar Amount") and Label 10 ("Total Alcohol Content"). The second plot shows incorrect and correct predictions colored in red and greed, respectively.



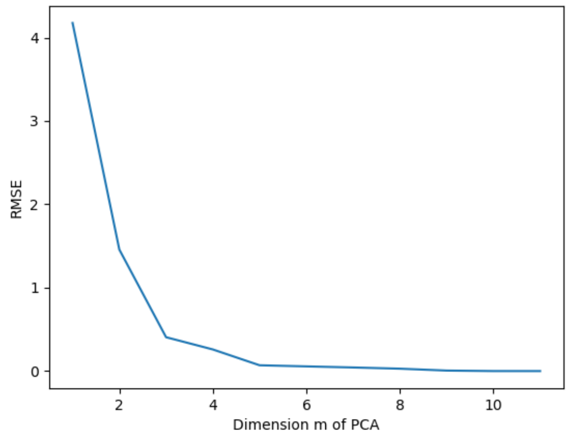


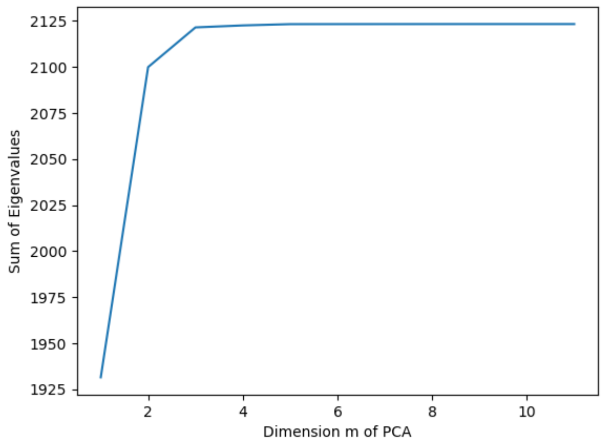
Next, dimensionality reduction was performed using PCA. A 3D projection can be seen below.

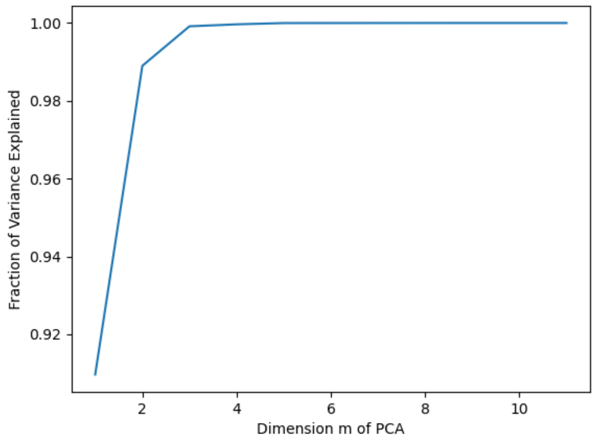


In the PCA projection into 3D space above, we can see that very few number of dimensions required to reach a RMSE value approaching 0. This gives the impression that PCA is an effective reduction technique. However, because the variance is already so high and concentrated, there is still a limited use-case for classifying this dataset. What we can see, however, in both this projection and the feature selection plot, that a gaussian distribution appears to match the data reasonably well.

The plots of Root-Mean-Squared-Error (RMSE), Sum of Eigenvalues, and Fraction of Variance Explained were plotted against the increasing dimensions (principle components) found through PCA. It can be seen that as more dimensions are used, the relative error decreases towards zero, the sum of eigenvalues increases, and the fraction of variance explained tends towards 1. Depending on the requirements of the problem, using 3 principle components (in this case) significantly reduces the error down to near that of using 10 dimensions! Therefore, PCA is a useful tool for this application. This also showcases that the assumption of a gaussian distribution is not appropriate for this dataset.







Human Activity Dataset

The Human Activity dataset has a total of 7351 training samples following the distribution below. From that, the priors were calculated. It can be seen that label 1 was never labeled, so (similar to the White Wine Dataset) there will never be any decisions made for those labels. Therefore, in the code attached to this report, all analysis was done by ignoring this first label, but for the sake of completeness, the confusion matrix has been expanded to include those empty labels.

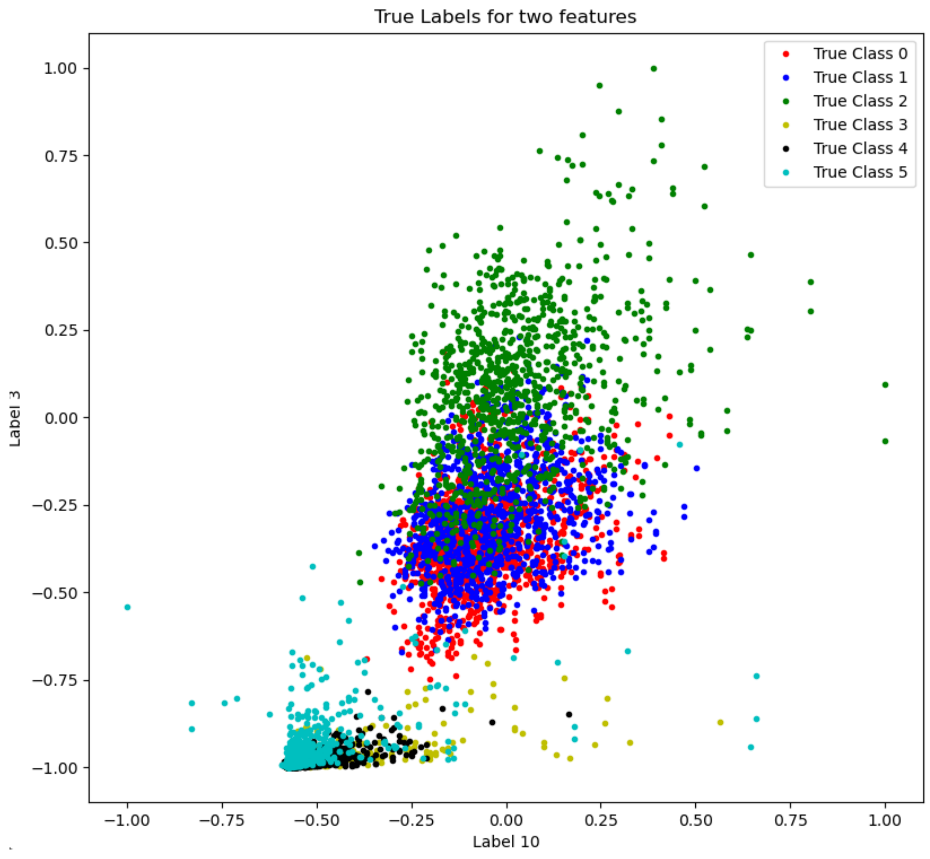
Label	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Num Samples	0	1226	1073	986	1286	1373	1407
Priors	0	0.1667	0.1459	0.1341	0.1749	0.1868	0.1914

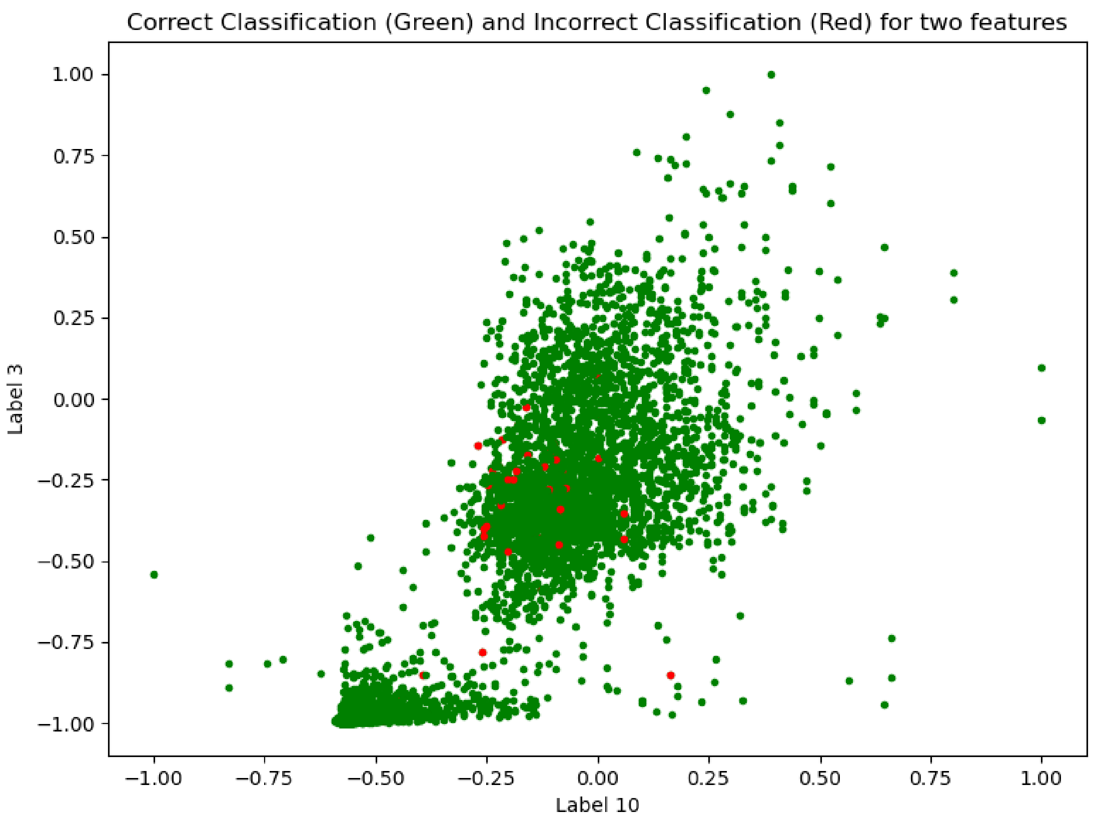
The total number of misclassifications was found to be 261, which corresponds to an error estimate of 0.035, which is a significant improvement over the white wine dataset (comparison in the conclusions section below).

The Confusion Matrix below shows the probabilities of correctly or incorrectly classifying samples.

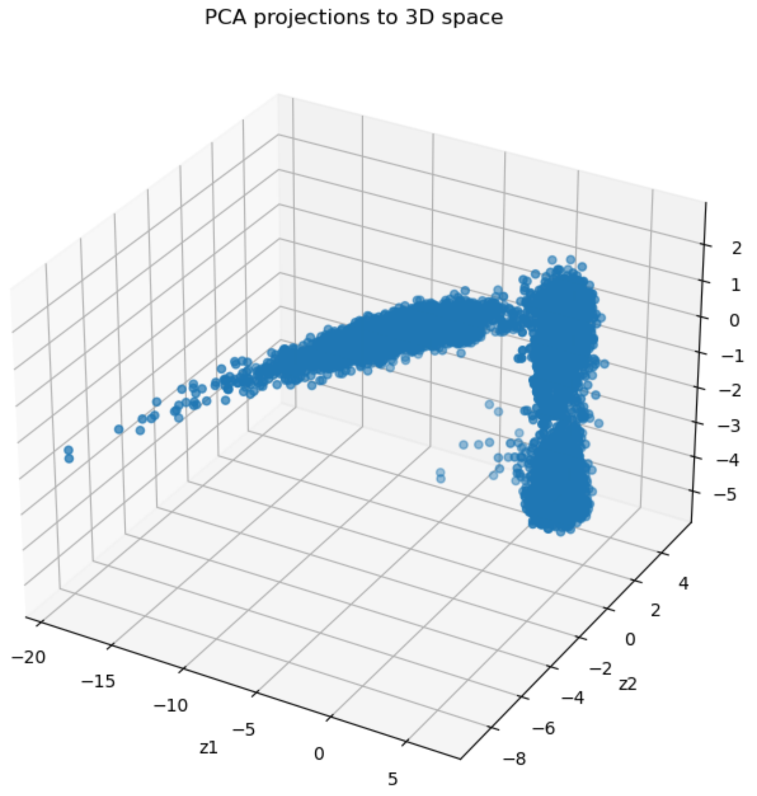
		Truth						
		1	**2**	**3**	**4**	**5**	**6**	**7**
Decision	**1**	0	0	0	0	0	0	0
	2	0	0.999	0	0.002	0	0	0
	3	0	0.001	1.000	0.030	0.001	0	0
	4	0	0	0	0.968	0	0	0
	5	0	0	0	0	0.827	0.004	0
	6	0	0	0	0	0.172	0.996	0
	7	0	0	0	0	0	0	1.000

The plots below show a visualization of the human activity dataset using a subset of features, which in this case is Label 3 ("Body Gyro") and Label 10 ("Body Gyro Jerk"). The second plot shows incorrect and correct predictions colored in red and greed, respectively.





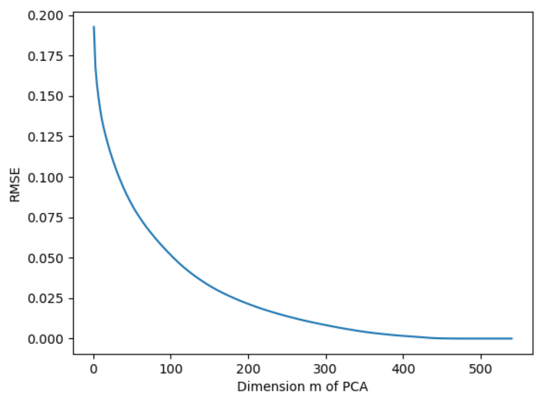
Next, dimensionality reduction was performed using PCA. A 3D projection can be seen below.

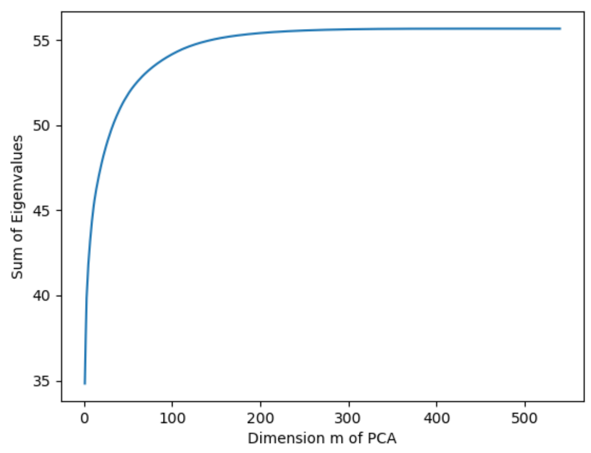


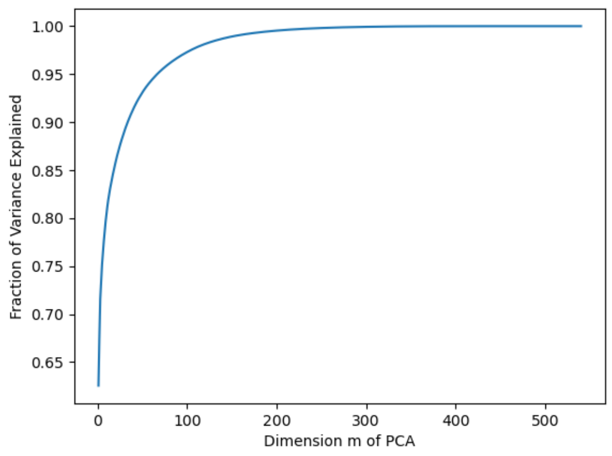
The PCA projection into 3D space is shown above. PCA is very useful when there are many features, and in this case there are over 500.

On another note, the data plotted from PCA and from feature subsets does not appear gaussian, but due to the large number of features and labels, using a gaussian estimation is reasonable for this dataset.

The plots of Root-Mean-Squared-Error (RMSE), Sum of Eigenvalues, and Fraction of Variance Explained were plotted against the increasing dimensions (principal components) found through PCA. It can be seen that as more dimensions are used, the relative error decreases towards zero, the sum of eigenvalues increases, and the fraction of variance explained tends towards 1. In this situation, PCA is can retain the majority of information using only half or fewer of the features.







Conclusions

This problem demonstrates the importance of data on the ability of an algorithm to classify samples. Wine tasting is well-known to be more of an art than a science, and even experts are making it up as they go. This is demonstrated in how difficult it was for our classification algorithms, even given true labels and information, to correctly create distinctions between wines. The human activity dataset, however, has significantly greater success because there are more clear correlations in sensor measurements (accelerometer, gyro, heart rate, etc.) to different exercises.

As discussed above, the assumption of a gaussian distribution for real-world data is not always applicable, but for both of these cases it is a reasonable assumption, with the white wine dataset being a challenging dataset to tackle regardless.

The choice was made to use the Bayesian Decision Theory (MAP) classifier to optimally minimize the probability of error. This resulted in a confusion matrix and a calculation of minP(error). Using other methods such as Naïve Bayes or Fisher LDA, as seen in problem 1, would result in a less optimal value when compared to the theoretically optimum.

In the comparison of these two datasets, it was fascinating to see how different the results can be. The white wine dataset has an error about half of the time, while the human activity dataset is very close to perfect classification. This makes sense because the human activity is directly correlated to a specific movement or activity, such as walking up stairs compared to being sedentary. For example, a high heart rate can easily be correlated with certain activities. However, wine is well-known for being very subjective and difficult to classify, even by experts. It was interesting to see that even a trained algorithm has trouble correctly identifying different wines. Of course, increasing the dataset size or number of features would benefit it greatly.

Appendix: Code / Github

The Code for each problem can be found in the Appendices below.

It can also be found in my github at https://github.com/meuliano/EECE5644/tree/main/HW1 with the following python scripts:

- Problem 1: Euliano_eece5644_hw1_1.py
- Problem 2: Euliano_eece5644_hw1_2.py
- Problem 3: Euliano_eece5644_hw1_3.py

Appendix A: Problem 1 Code

```
# %%
# Intro to Machine Learning and Pattern Recognition - EECE5644
# Homework 1 - Problem 1
# Author: Matthew Euliano
import matplotlib.pyplot as plt #General Plotting
from matplotlib import cm
import numpy as np
from scipy.stats import multivariate_normal
from sys import float_info # Threshold smallest positive floating value

# Set seed to generate reproducible "pseudo-randomness" (handles scipy's
"randomness" too)
np.random.seed(4)



# %%
def main():
    N = 10000 # Samples

    # Means and Covariance Matrices
    mu = np.array([[-1/2, -1/2, -1/2], [1, 1, 1]])
    sigma = np.array([[[1,-0.5,0.3],[-0.5,1,-0.5],[0.3, -0.5, 1]],
                    [[1, 0.3, -0.2],[0.3, 1, 0.3],[-0.2, 0.3, 1]]])

    priors = np.array([0.65, 0.35]) # Given Priors

    n = mu.shape[1] # Sample dimensions (3)
    C = len(priors) # Number of Classes (2)
    L = np.array(range(C)) # Define labels as [0 1]

    # set thresholds: results in [0.0, 0.65, 1.0]
    thresholds = np.cumsum(priors)
    thresholds = np.insert(thresholds,0,0)

    X = np.zeros([N,n]) # create empty matrix for samples of size 10,000 x 3
    labels = np.zeros(N) # create empty labels array of length 10,000

    # Create a 10000-length vector of uniformly distributed random variables
in [0,1)
    # Will be used to get an estimate of number of features belonging to each
label
    u = np.random.rand(N)
    # Plot for original data and their true labels
    fig = plt.figure(figsize=(12, 10))
    ax1 = fig.add_subplot(131, projection = '3d')
    marker_shapes = '..'
    marker_colors = 'rb'

    for i in range(C):

        # Find indices of u that meet each prior
        indices = np.argwhere((thresholds[i] <= u) & (u <=
thresholds[i+1]))[:, 0]

        Nl = len(indices) # Get number of indices in each component: should
be ~ 6500 and 3500

        # set label vector based on above - will be vector of class labels [0
0 1 0 1 .. to 9999] in this case
        labels[indices] = i * np.ones(Nl)

        # for each valid index, insert 3D random sample using relevant
gaussian distribution
        X[indices, :] = multivariate_normal.rvs(mu[i], sigma[i], Nl)

        # Add to scatter plot
        ax1.scatter(X[labels==i, 0], X[labels==i, 1], X[labels==i, 2],
marker=marker_shapes[i], c=marker_colors[i], label="True Class {}".format(i))

    plt.legend()
    ax1.set_xlabel(r"$x_1$")
    ax1.set_ylabel(r"$x_2$")
    ax1.set_zlabel(r"$x_3$")
    plt.title("Generated Original Data Samples")
    plt.tight_layout()
    plt.show()

    # Generate data distributions and store in true class labels
    Nl = np.array([sum(labels == i) for i in range(C)])
    print("Num samples: 0 = {}, 1 = {}".format(Nl[0],Nl[1]))


    # %%
    # Generate ROC curve samples
    def estimate_roc(discriminant_score, label):
        Nlabels = np.array((sum(label == 0), sum(label == 1))) #~[6500,3500]

        sorted_score = sorted(discriminant_score)

        # Use tau values that will account for every possible classification
split
        # These are all the possible values of Gamma that we will test
through
        taus = ([sorted_score[0] - float_info.epsilon] + sorted_score +
                [sorted_score[-1] + float_info.epsilon])

        # Calculate the decision label for each observation for each gamma
[10000]
```

```
        decisions = [discriminant_score >= t for t in taus]

        # True Positive
        ind11 = [np.argwhere((d==1) & (label==1)) for d in decisions]
        p11 = [len(inds)/Nlabels[1] for inds in ind11]
        # False Positive
        ind10 = [np.argwhere((d==1) & (label==0)) for d in decisions]
        p10 = [len(inds)/Nlabels[0] for inds in ind10]

        # ROC has FPR on the x-axis and TPR on the y-axis
        roc = np.array((p10, p11))

        # Calculate error probability
        prob_error = [(p10[w] *priors[0] + (1 - p11[w])* priors[1]) for w in
range(len(p10))]

        return roc, taus, prob_error

    # %%
    # Expected Risk Minimization Classifier - Estimated

    # Class conditional likelihood is a 2x10,000 matrix of likelihoods for
each sample based on gaussian distribution
    class_conditional_likelihoods = np.array([multivariate_normal.pdf(X,
mu[l], sigma[l]) for l in L])

    # Discriminant Score using log-likelihood ratio: is a 10,000-length
vector of values to compare to the threshold
    discriminant_score_erm = np.log(class_conditional_likelihoods[1]) -
np.log(class_conditional_likelihoods[0])

    # Vary Gamma gradually and compute True-Positive and False-Positive
probabilities
    roc_erm, tau, prob_error = estimate_roc(discriminant_score_erm, labels)

    # Find minimum error and index
    minimum_error = min(prob_error)
    minimum_index = prob_error.index(minimum_error)

    # Experimental / approximate Threshold Gamma value
    # e^ (undo log in prev cell)
    gamma_approx = np.exp(tau[minimum_index])
    print("Approximated Threshold: ", gamma_approx)
    print("Approximated Minimum Error: ", minimum_error)

    # %%
    # Minimum Expected Risk Classification using Likelihood-Ratio Test
    Lambda = np.ones((C, C)) - np.identity(C)
    # Theoretical / Expected
    # Gamma threshold for MAP decision rule (remove Lambdas and you obtain
same gamma on priors only; 0-1 loss simplification)
    gamma_th = (Lambda[1,0] - Lambda[0,0]) / (Lambda[0,1] - Lambda[1,1]) *
priors[0]/priors[1]
    # Same as: gamma_th = priors[0]/priors[1]
    print("Theoretical Gamma / Threshold = ", gamma_th)

    # get decision for EACH sample based on theoretically optimal threshold
    decisions_map = discriminant_score_erm >= np.log(gamma_th)

    # Get indices and probability estimates of the four decision scenarios:
    # (true negative, false positive, false negative, true positive)

    # True Negative Probability
    ind_00_map = np.argwhere((decisions_map==0) & (labels==0))
    p_00_map = len(ind_00_map) / Nl[0]
    # False Positive Probability
    ind_10_map = np.argwhere((decisions_map==1) & (labels==0))
    p_10_map = len(ind_10_map) / Nl[0]
    # False Negative Probability
    ind_01_map = np.argwhere((decisions_map==0) & (labels==1))
    p_01_map = len(ind_01_map) / Nl[1]
    # True Positive Probability
    ind_11_map = np.argwhere((decisions_map==1) & (labels==1))
    p_11_map = len(ind_11_map) / Nl[1]

    # Theoretical
    roc_map = np.array((p_10_map, p_11_map))

    # Probability of error for MAP classifier, empirically estimated
    prob_error_th = (p_10_map *priors[0] + (1 - p_11_map)* priors[1])
    print("Theoretical Minimum Error: ", prob_error_th)



    # %%
    # Plot ROC
    fig_roc, ax_roc = plt.subplots(figsize=(10, 10))
    ax_roc.plot(roc_erm[0], roc_erm[1])
    ax_roc.plot(roc_erm[0,minimum_index], roc_erm[1,minimum_index],'b+',
label="Experimental Minimum P(Error)", markersize=16)
    ax_roc.plot(roc_map[0], roc_map[1], 'rx', label="Theoretical Minimum
P(Error)", markersize=16)
    ax_roc.legend()
    ax_roc.set_xlabel(r"Probability of false alarm $P(D=1|L=0)$")
    ax_roc.set_ylabel(r"Probability of correct decision $P(D=1|L=1)$")
    plt.title("Minimum Expected Risk ROC Curve - ERM")
    plt.grid(True)

    fig_roc;


    # Display MAP decisions
    fig = plt.figure(figsize=(10, 10))
    ax4 = fig.add_subplot(131, projection = '3d')

    # class 0 circle, class 1 +, correct green, incorrect red
    ax4.scatter(X[ind_00_map, 0], X[ind_00_map, 1], X[ind_00_map, 2],
c='g',marker='o', label="Correct Class 0")
    ax4.scatter(X[ind_10_map, 0], X[ind_10_map, 1], X[ind_10_map, 2],
c='r',marker='o', label="Incorrect Class 0")
    ax4.scatter(X[ind_01_map, 0], X[ind_01_map, 1], X[ind_01_map, 2],
c='r',marker='+', label="Incorrect Class 1")
    ax4.scatter(X[ind_11_map, 0], X[ind_11_map, 1], X[ind_11_map, 2],
c='g',marker='+', label="Correct Class 1")
```

```
    plt.legend()
    plt.xlabel(r"$x_1$")
    plt.ylabel(r"$x_2$")
    plt.title("MAP Decisions (RED incorrect)")
    plt.tight_layout()
    plt.show()

    # %%
    # PART B
    # Naive Bayes Sigma value assumes incorrect covariance matrices setting
off-diag vals to 0
    sigma_nb = np.array([[1, 0, 0],
                        [0, 1, 0],
                        [0, 0, 1]])

    class_conditional_likelihoods_nb = np.array([multivariate_normal.pdf(X,
mu[l], sigma_nb) for l in L])
    discriminant_score_erm_nb = np.log(class_conditional_likelihoods_nb[1]) -
np.log(class_conditional_likelihoods_nb[0])

    roc_erm_nb, tau_nb, prob_error_nb =
estimate_roc(discriminant_score_erm_nb, labels)

    # Find minimum error and index
    minimum_error_nb = min(prob_error_nb)
    print("Minimum Error = ", minimum_error_nb)
    minimum_index_nb = prob_error_nb.index(minimum_error_nb)

    # Experimental / approximate Threshold Gamma value
    # e^ (undo log in prev cell)
    gamma_approx_nb = np.exp(tau_nb[minimum_index_nb])
    print("Approximated Threshold = ", gamma_approx_nb)


    # %%
    # Plot ROC
    fig_roc_nb, ax_roc_nb = plt.subplots(figsize=(10, 10))
    ax_roc_nb.plot(roc_erm_nb[0], roc_erm_nb[1])
    ax_roc_nb.plot(roc_erm_nb[0,minimum_index_nb],
roc_erm_nb[1,minimum_index_nb],'b+', label="Experimental Minimum P(Error)",
markersize=16)
    # ax_roc.plot(roc_map[0], roc_map[1], 'rx', label="Theoretical Minimum
P(Error)", markersize=16)
    ax_roc_nb.legend()
    ax_roc_nb.set_xlabel(r"Probability of false alarm $P(D=1|L=0)$")
    ax_roc_nb.set_ylabel(r"Probability of correct decision $P(D=1|L=1)$")
    plt.grid(True)
    plt.title("Minimum Expected Risk ROC Curve - Naive Bayes")
    plt.show()

    fig_roc_nb;

    # %%
    # PART C
    def perform_lda(X, mu, Sigma, C=2):
        """  Fisher's Linear Discriminant Analysis (LDA) on data from two
classes (C=2).

        In practice the mean and covariance parameters would be estimated
from training samples.

        Args:
            X: Real-valued matrix of samples with shape [N, n], N for sample
count and n for dimensionality.
            mu: Mean vector [C, n].
            Sigma: Covariance matrices [C, n, n].

        Returns:
            w: Fisher's LDA project vector, shape [n, 1].
            z: Scalar LDA projections of input samples, shape [N, 1].
        """

        mu = np.array([mu[i].reshape(-1, 1) for i in range(C)])
        cov = np.array([Sigma[i].T for i in range(C)])

        # Determine between class and within class scatter matrix
        Sb = (mu[0] - mu[1]).dot((mu[0] - mu[1]).T)
        Sw = cov[0] + cov[1]

        # Regular eigenvector problem for matrix Sw^-1 Sb
        lambdas, U = np.linalg.eig(np.linalg.inv(Sw).dot(Sb))
        # Get the indices from sorting lambdas in order of increasing value,
with ::-1 slicing to then reverse order
        idx = lambdas.argsort()[::-1]

        # Extract corresponding sorted eigenvectors
        U = U[:, idx]

        # First eigenvector is now associated with the maximum eigenvalue,
mean it is our LDA solution weight vector
        w = U[:, 0]

        # Scalar LDA projections in matrix form
        z = X.dot(w)

        return w, z

    # %%
    # calculate mu projections
    mu0proj = np.array([np.average(X[labels==0][:,i]) for i in range(n)]).T
    mu1proj = np.array([np.average(X[labels==1][:,i]) for i in range(n)]).T
    mu_proj = np.array([mu0proj, mu1proj])

    sigma0proj = np.cov(X[labels==0], rowvar=False)
    sigma1proj = np.cov(X[labels==1], rowvar=False)
    sigma_proj = np.array([sigma0proj, sigma1proj])

    # Fisher LDA Classifer (using true model parameters)
    w, discriminant_score_lda = perform_lda(X, mu_proj, sigma_proj)

    # Estimate the ROC curve for this LDA classifier
    roc_lda, tau_lda, prob_error_lda = estimate_roc(discriminant_score_lda,
labels)

    minimum_error_lda = min(prob_error_lda)
    minimum_index_lda = prob_error_lda.index(minimum_error_lda)

    # Experimental / approximate Threshold Gamma value
    # e^ (undo log in prev cell)
    gamma_approx_lda = np.exp(tau_lda[minimum_index_lda])
    print("Approximated LDA Threshold: ", gamma_approx_lda)
    print("Approximated LDA Minimum Error: ", minimum_error_lda)

    fig_roc_lda, ax_roc_lda = plt.subplots(figsize=(10, 10))
    ax_roc_lda.plot(roc_lda[0], roc_lda[1])
    ax_roc_lda.plot(roc_lda[0,minimum_index_lda],
roc_lda[1,minimum_index_lda],'g+', label="Experimental Minimum P(Error)",
markersize=16)
    # ax_roc_lda.plot(roc_map[0], roc_map[1], 'rx', label="Theoretical
Minimum P(Error)", markersize=16)
    ax_roc_lda.legend()
    ax_roc_lda.set_xlabel(r"Probability of false alarm $P(D=1|L=0)$")
    ax_roc_lda.set_ylabel(r"Probability of correct decision $P(D=1|L=1)$")
    plt.title("Minimum Expected Risk ROC Curve - LDA")
    plt.grid(True)
    plt.show()

    fig_roc_lda;

    # %%

    fig_roc_all, ax_roc_all = plt.subplots(figsize=(10, 10))
    ax_roc_all.plot(roc_erm[0], roc_erm[1])
    ax_roc_all.plot(roc_erm[0,minimum_index], roc_erm[1,minimum_index],'b.',
label="ERM P(Error)", markersize=16)
    ax_roc_all.plot(roc_map[0], roc_map[1], 'gx', label="Theoretical Minimum
P(Error)", markersize=16)
    ax_roc_all.plot(roc_erm_nb[0], roc_erm_nb[1])
    ax_roc_all.plot(roc_erm_nb[0,minimum_index_nb],
roc_erm_nb[1,minimum_index_nb],'r.', label="Naive Bayes P(Error)",
markersize=16)
    ax_roc_all.plot(roc_lda[0], roc_lda[1])
    ax_roc_all.plot(roc_lda[0,minimum_index_lda],
roc_lda[1,minimum_index_lda],'m.', label="Fisher LDA P(Error)",
markersize=16)
    # ax_roc_lda.plot(roc_map[0], roc_map[1], 'rx', label="Theoretical
Minimum P(Error)", markersize=16)
    ax_roc_all.legend()
    ax_roc_all.set_xlabel(r"Probability of false alarm $P(D=1|L=0)$")
    ax_roc_all.set_ylabel(r"Probability of correct decision $P(D=1|L=1)$")
    plt.title("Minimum Expected Risk ROC Curve - ALL")
    plt.grid(True)
    plt.show()


if __name__ == '__main__':
    main()
```

Appendix B: Problem 2 Code

```
# Intro to Machine Learning and Pattern Recognition - EECE5644
# Homework 1 - Problem 2
# Author: Matthew Euliano
import matplotlib.pyplot as plt #General Plotting
import numpy as np
from scipy.stats import multivariate_normal


# Set seed to generate reproducible "pseudo-randomness" (handles scipy's
"randomness" too)
np.random.seed(4)

N = 10000 # samples

def main():
    priors = np.array([.2,.25,.25,.3]) # Given Priors

    # ARBITRARILY CHOSEN mean and covariance matrix
    # equally spaced means along a line
    mu =np.array([[0, 0],[3, 0],[6, 0],[9, 0]])

    # scaled versions of the identity matrix with overlap
    sigma =[[[2, 0],[0, 2]],
            [[4, 0],[0, 4]],
            [[6, 0],[0, 6]],
            [[8, 0],[0, 8]]]

    n = mu.shape[1] # sample dimensions (2)
    C = len(priors) # Classes (4)
    L = np.array(range(C)) # labels [0 1 2 3]
    Y = np.array(range(C))  # 0-(C-1)

    # set thresholds: results in [0.0, 0.2, 0.45, 0.7, 1.0]
    thresholds = np.cumsum(priors)
    thresholds = np.insert(thresholds,0,0)

    X = np.zeros([N,n]) # create empty matrix for samples of size 10,000 x 3
    labels = np.zeros(N) # create empty labels array of length 10,000

    # Create a 10000-length vector of uniformly distributed random variables
in [0,1)
    # Will be used to get an estimate of number of features belonging to each
label
    u = np.random.rand(N)

    # Plot for original data and their true labels
    fig = plt.figure(figsize=(10, 10))
    marker_shapes = '....'
    marker_colors = 'rbgy'

    for i in range(C):

        # Find indices of u that meet each prior
        indices = np.argwhere((thresholds[i] <= u) & (u <=
thresholds[i+1]))[:, 0]
```

```
        # Get the number of indices in each component - should be ~ 6500 and
3500
        Nl = len(indices)

        # set label vector based on above - will be vector of class labels [0
0 1 0 1 .. to 9999] in this case
        # for more classes, this can be [1 2 2 0 1 3 1 3 0 ...]
        labels[indices] = i * np.ones(Nl)

        # for each valid index...
        X[indices, :] = multivariate_normal.rvs(mu[i], sigma[i], Nl)
        plt.plot(X[labels==i, 0], X[labels==i, 1], marker_shapes[i-1] +
marker_colors[i-1], label="True Class {}".format(i))

    # Plot the original data and their true labels
    plt.legend()
    plt.xlabel(r"$x_1$")
    plt.ylabel(r"$x_2$")
    plt.title("Generated Original Data Samples")
    plt.tight_layout()
    plt.show()

    # Generate data distributions and store in true class labels
    Nl = np.array([sum(labels == i) for i in range(C)])
    print("Num samples: 0 = {}, 1 = {}, 2 = {}, 3 =
{}".format(Nl[0],Nl[1],Nl[2],Nl[3]))

    # Lambda Matrix
    lambda_matrix = np.ones((C, C)) - np.identity(C)

    # Calculate class-conditional likelihoods p(x|Y=j) for each label of the
N observations
    class_cond_likelihoods = np.array([multivariate_normal.pdf(X, mu[j],
sigma[j]) for j in Y])
    class_priors = np.diag(priors)
    class_posteriors = class_priors.dot(class_cond_likelihoods)

    # We want to create the risk matrix of size 3 x N
    cond_risk = lambda_matrix.dot(class_posteriors)

    # Get the decision for each column in risk_mat
    decisions = np.argmin(cond_risk, axis=0)

    # Plot for decisions vs true labels
    fig = plt.figure(figsize=(12, 10))
    marker_shapes = 'ox+*.' # Accomodates up to C=5
    marker_colors = 'brgmy'

    # Get sample class counts
    sample_class_counts = np.array([sum(labels == j) for j in Y])

    # Confusion matrix
    conf_mat = np.zeros((C, C))
    for i in Y: # Each decision option
        for j in Y: # Each class label
            ind_ij = np.argwhere((decisions==i) & (labels==j))
            conf_mat[i, j] = round(len(ind_ij)/sample_class_counts[j],3) # 
Average over class sample count
            if i == j:
                # True label = Marker shape; Decision = Marker Color
                marker = marker_shapes[j] + marker_colors[i]
                plt.plot(X[ind_ij, 0], X[ind_ij, 1], 'g'+marker_shapes[j], 
markersize=6,label="Correct decision {}".format(i))
            else:
                plt.plot(X[ind_ij, 0], X[ind_ij, 1], 'r'+marker_shapes[j], 
markersize=6,label="Incorrect Decision {} in label {}".format(i,j))

    print("Confusion matrix:")
    print(conf_mat)

    print("Minimum Probability of Error:")
    prob_error = 1 - np.diag(conf_mat).dot(sample_class_counts / N)
    print(prob_error)

    plt.legend()
    plt.title("Minimum Probability of Error Classified Sampled Data: 
{:.3f}".format(prob_error))
    plt.xlabel(r"$x_1$")
    plt.ylabel(r"$x_2$")
    plt.tight_layout()
    plt.show()

    # %%
    # Part B
    lambda_matrix_b =np.array([[ 0, 1, 2, 3],[1, 0, 1, 2],[2, 1, 0, 1],[3, 2, 
1, 0]])

    # We want to create the risk matrix of size 3 x N
    cond_risk_b = lambda_matrix_b.dot(class_posteriors)

    # Get the decision for each column in risk_mat
    decisions_b = np.argmin(cond_risk_b, axis=0)
    print(decisions_b)

    # Plot for decisions vs true labels
    fig = plt.figure(figsize=(12, 10))
    marker_shapes = 'ox+*.' # Accomodates up to C=5
    marker_colors = 'brgmy'

    # Get sample class counts
    sample_class_counts = np.array([sum(labels == j) for j in Y])

    # Confusion matrix
    conf_mat_b = np.zeros((C, C))
    for i in Y: # Each decision option
        for j in Y: # Each class label
            ind_ij = np.argwhere((decisions_b==i) & (labels==j))
            conf_mat_b[i, j] = round(len(ind_ij)/sample_class_counts[j],3) # 
Average over class sample count

            if i == j:
                # True label = Marker shape; Decision = Marker Color
                marker = marker_shapes[j] + marker_colors[i]
                plt.plot(X[ind_ij, 0], X[ind_ij, 1], 'g'+marker_shapes[j],
markersize=6,label="Correct decision {}".format(i))

            else:
                plt.plot(X[ind_ij, 0], X[ind_ij, 1], 'r'+marker_shapes[j],
markersize=6,label="Incorrect Decision {} in label {}".format(i,j))

    print("Confusion matrix:")
    print(conf_mat_b)

    print("Minimum Probability of Error:")
    prob_error_b = 1 - np.diag(conf_mat_b).dot(sample_class_counts / N)
    print(prob_error_b)

    plt.legend()
    plt.title("Minimum Probability of Error Classified Sampled Data:
{:.3f}".format(prob_error_b))
    plt.xlabel(r"$x_1$")
    plt.ylabel(r"$x_2$")
    plt.tight_layout()
    plt.show()



if __name__ == '__main__':
    main()
```

Appendix C: Problem 3 Code

```
# %%
# Intro to Machine Learning and Pattern Recognition - EECE5644
# Homework 1 - Problem 3
# Author: Matthew Euliano
from pandas import read_csv
import matplotlib.pyplot as plt #General Plotting
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.metrics import confusion_matrix
import os
script_dir = os.path.abspath( os.path.dirname( __file__ ) )

# %% Implement minimum-probability-of-error classifier
def erm_classifier(X,valid_classes,labels):
    N = len(labels)
    print("Num samples: ", N)

    N_cl = np.array([sum(labels == i) for i in valid_classes])
    print("Num Class Labels: ", N_cl)

    priors = np.array(N_cl/N)
    print("Priors: ", priors)

    C = len(priors)

    # First derive sample-based estimates of mean vector and covariance
matrix:
    mu_hat = np.array([np.mean(X[labels == i], axis = 0) for i in
valid_classes]) # No labelled samples for 0, 1, 2, 10!

    reg = 0.1*np.identity(X.shape[1]) #7x11

    Sigma_hat = np.array([np.cov(X[labels == i].T) + reg for i in
valid_classes]) #7x11x11

    # Implement minimum P(error) classification rule on all training samples
    class_cond_likelihoods = np.array([multivariate_normal.pdf(X, mu_hat[c],
Sigma_hat[c]) for c in range(len(valid_classes))])
    priors_diag = np.diag(priors)
    class_posteriors = priors_diag.dot(class_cond_likelihoods)

    decisions = np.argmax(class_posteriors, axis = 0)
    decisions = np.array([i + valid_classes[0] for i in decisions]) # labels
start at 3...

    conf_matrix = confusion_matrix(decisions, labels)
    print("Confusion Matrix:\n",conf_matrix)

    errors = len(np.argwhere(decisions != labels))
    print('Number of Misclassifications: ', errors, "\nError Estimate: ",
errors/N)

    # Confusion matrix
    conf_mat_b = np.zeros((C, C))
    for i in valid_classes: # Each decision option
        for j in valid_classes: # Each class label
            ind_ij = np.argwhere((decisions==i) & (labels==j))
            conf_mat_b[i, j] = round(len(ind_ij)/N_cl[j],3) # Average over
class sample count

            # True label = Marker shape; Decision = Marker Color
            plt.plot(X[ind_ij, 10], X[ind_ij, 6], 'g.', markersize=6)

            if i != j:
                plt.plot(X[ind_ij, 10], X[ind_ij, 6], 'r.', markersize=6)

    print(conf_mat_b)
    plt.title("Correct Classification (Green) and Incorrect Classification
(Red) for two features")
    plt.xlabel(r"Label 10")
    plt.ylabel(r"Label 3")
    plt.tight_layout()
    plt.show()

    # Plot for original data and their true labels
    fig = plt.figure(figsize=(10, 10))
    marker_shapes = '.......'
    marker_colors = 'rbgykcm'

    for i in valid_classes:
        plt.plot(X[labels==i, 10], X[labels==i, 6], marker_shapes[i] +
marker_colors[i], label="True Class {}".format(i))

    plt.legend()
    plt.title("True Labels for two features")
    plt.xlabel(r"Label 10")
    plt.ylabel(r"Label 3")
    plt.tight_layout()
    plt.show()

# %% Implement PCA Dimensionality Reduction
def pca_reducttion(X):
    # Part B - PCA
    # First derive sample-based estimates of mean vector and covariance
matrix:
    mu_hat = np.mean(X, axis=0)
    Sigma_hat = np.cov(X.T)

    # Mean-subtraction is a necessary assumption for PCA, so perform this to
obtain zero-mean sample set
    C = X - mu_hat

    # Get the eigenvectors (in U) and eigenvalues (in D) of the estimated
covariance matrix
    lambdas, U = np.linalg.eig(Sigma_hat)
    # Get the indices from sorting lambdas in order of increasing value, with
::-1 slicing to then reverse order
    idx = lambdas.argsort()[::-1]
    # Extract corresponding sorted eigenvectors and eigenvalues
    U = U[:, idx]
    D = np.diag(lambdas[idx])
    # Calculate the PC projections of zero-mean samples (in z)
    Z = np.real(C.dot(U))

    # Let's see what it looks like only along the first two PCs
    fig = plt.figure(figsize=(10, 10))
    ax1 = fig.add_subplot(111, projection = '3d')
    ax1.scatter(Z[:, 0], Z[:, 1], Z[:,2])
    plt.xlabel("z1")
    plt.ylabel("z2")
    plt.title("PCA projections to 3D space")
    plt.show()


    # Max number of PCs based on rank of X, or min(n, N)
    X_rank = np.linalg.matrix_rank(X)

    rmse = np.zeros(X_rank)
    sum_eigenvals = np.zeros(X_rank)
    no_components = range(1, X_rank + 1)

    # Reconstruct the X data set from each set of projections
    for m in no_components:
        # Reconstruct based on only the 'm' components (also revert mean-
centering effect)
        X_hat = Z[:, :m].dot(U[:, :m].T) + mu_hat
        rmse[m-1] = np.real(np.sqrt(np.mean((X - X_hat)**2)))
        sum_eigenvals[m-1] = np.real(np.sum(D[:m]))

    # Fraction of variance explained
    fraction_var = sum_eigenvals / np.trace(Sigma_hat)

    # MSE should be decreasing on each iteration, 0 for the nth
    plt.figure(1)
    plt.plot(no_components, rmse)
    plt.xlabel("Dimension m of PCA")
    plt.ylabel("RMSE")
    plt.show()

    # First eigenvalue should be significantly larger than the rest
    plt.figure(2)
    plt.plot(no_components, np.real(sum_eigenvals))
    plt.xlabel("Dimension m of PCA")
    plt.ylabel("Sum of Eigenvalues")
    plt.show()

    # About 95% variance explined is an acceptable target
    plt.figure(3)
    plt.plot(no_components, fraction_var)
    plt.xlabel("Dimension m of PCA")
    plt.ylabel("Fraction of Variance Explained")
    plt.show()


def run_wine_dataset():
    # fixed acidity;"volatile acidity";"citric acid";"residual
sugar";"chlorides";"free sulfur dioxide";"total sulfur
dioxide";"density";"pH";"sulphates";"alcohol";"quality"
    winedata = np.array(read_csv(script_dir + "\winequality-white.csv", sep =
';'))

    # labels 0 - 3 are empty, remove them from the list
    winedata[:,-1] = np.array([winedata[i,-1] -3 for i in
range(winedata.shape[0])])
    labels = winedata[:,-1] # take provided labels from last column of
dataset

    valid_classes = np.array([0,1,2,3,4,5,6])
    print("Valid Classes: ", valid_classes)

    X = np.array(winedata[:,0:11])

    erm_classifier(X,valid_classes,labels)
    pca_reducttion(X)

# %% Run Human Activity Dataset
def run_activity_dataset():
    # Activity Data
    # Import Human Activity Dataset
    activitydata = np.array(read_csv(script_dir + "\X_train.txt", sep =
'\s+'))

    # Import Human Activity Labels (separate txt file) as a vector
    labels = np.array(read_csv(script_dir + "\y_train.txt", sep =
'\s+')).T[0]
    # Remove the first empty column
    labels = np.array([i - 1 for i in labels])

    valid_classes = np.array(range(6))
    print(valid_classes)

    X = activitydata

    erm_classifier(X,valid_classes,labels)
    pca_reducttion(X)

# %%
if __name__ == '__main__':

    run_wine_dataset()
    run_activity_dataset()
```