DATA/STAT/BIOSTAT 558
SPRING QUARTER 2025

Homework # 2
Online Submission to Canvas: due Wednesday April 30th, 5pm PST

Instructions: You may discuss the homework problems in small groups, but you must write up the final solutions and code yourself. Please turn in your code used for the problems that involve coding. However, code without written answers will receive no credit. To receive credit, you must explain your answers and show your work. All plots should be appropriately labeled and legible, with axis labels, legends, etc., as needed.

1. Suppose we have a quantitative response Y, and two features X_1 and X_2. Let RSS_1 denote the residual sum of squares that results from fitting the model

$$Y = \beta_0 + \beta_1 X_1 + \epsilon$$

using least squares. Let RSS_{12} denote the residual sum of squares that results from fitting the model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

using least squares.

(a) Argue that $\text{RSS}_{12} \leq \text{RSS}_1$.
Solution : The least squares model always chooses coefficient estimates that minimize the RSS for the given model.
Model 1 (with RSS_1) : $Y = \beta_0 + \beta_1 X_1 + \epsilon$
Model 2 (with RSS_{12}) : $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$

Model 2 includes all the predictors in Model 1 and will try to get a better fit as it has an additional feature. When fitting Model 2, the least squares procedure can always choose $\beta_2 = 0$ and get the same result as Model 1. Therefore, the model 2 will never produce a higher RSS than model 1.
Since Model 2 can mimic Model 1 but also has the potential to do better by choosing a non-zero β_2 we conclude $\text{RSS}_{12} \leq \text{RSS}_1$.

(b) For a fitted model $\hat{f}$ trained on data $(x_1, y_1), \ldots, (x_n, y_n)$, we define the R^2 (*proportion of variance explained*) of the fitted model to be

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. Using your result from 1(a), argue that the R^2 of the model containing just the feature X_1 is no greater than the R^2 of the model containing both X_1 and X_2.

Solution : $R^2 = 1 - \dfrac{RSS}{TSS}$

where TSS is $\sum_{i=1}^{n}(y_i - \bar{y})^2$ which will be same for both the models.

$$\begin{aligned} &\text{From part 1a we get } \mathrm{RSS}_{12} \leq \mathrm{RSS}_1 \\ \implies &\frac{\mathrm{RSS}_{12}}{\mathrm{TSS}} \leq \frac{\mathrm{RSS}_1}{\mathrm{TSS}} \\ \implies &1 - \frac{\mathrm{RSS}_{12}}{\mathrm{TSS}} \geq 1 - \frac{\mathrm{RSS}_1}{\mathrm{TSS}} \\ \implies &R^2_{12} \geq R^2_1 \end{aligned}$$

Hence R^2 of the model containing just the feature X_1 is no greater than the R^2 of the model containing both X_1 and X_2.

(c) If you used R^2 as the singular metric to evaluate the quality of your model, then the result in 1(b) suggests that you should add as many predictor variables as you possibly can to your model so that R^2 can be as high as possible. Why might this not be a good idea?

Solution : Using R^2 as the singular metric to evaluate model quality and aiming to maximize it by adding as many predictors as possible is not a good idea for several important reasons:

- **Overfitting**: R^2 will always increase when more variables are added to the model, even if those variables are only weakly associated with the response. This means that even irrelevant variables will seem to improve the model leading to overfitting. An overfit model will not perform well on independent test data.
- **Multicollinearity**: Adding more predictors which might be correlated can make the estimates of the coefficients unstable. This reduces the reliability of the model and can make the model sensitive to small changes in the input data.
- **Small increase**: After a certain point, each new variable may only increase R^2 by a tiny amount which might not justify the added complexity or the risk of introducing noise into the model.

(d) Design a simulation to empirically verify your result in 1(b). For $i = 1, 2, \ldots, 200$, draw $X_i \overset{\text{i.i.d.}}{\sim} \text{Normal}(0, 3^2)$ and $Z_i \overset{\text{i.i.d.}}{\sim} \text{Exponential}(4)$, and then set

$$Y = 2 - 3X_i + \epsilon_i \tag{1}$$

where $\epsilon_i \overset{\text{i.i.d.}}{\sim} \text{Normal}(0, 2^2)$. Then, fit the following three models using least squares:

$$
\begin{aligned}
Y_i &= \beta_0 + \beta_1 X_i + \epsilon_i \\
Y_i &= \beta_0 + \beta_1 X_i + \beta_2 Z_i + \epsilon_i \\
Y_i &= \beta_0 + \beta_1 X_i + \beta_2 Z_i + \beta_3 \sin(X_i) + \epsilon_i
\end{aligned}
$$

Calculate the resulting R^2 for each of these three models, repeating the above simulation $b = 1, 2, \ldots, 100$ times. Using your own creativity, create a visual that demonstrates that the R^2 from the models with more predictors is never smaller than the models with less predictors. Importantly, note that this problem does not ever involve any "test data," you are only using training data for this problem.

(*For example, you could label the resulting R^2 values from the three models as $R^2_{1,b}$, $R^2_{2,b}$, and $R^2_{3,b}$, then you could calculate $R^2_{2,b} - R^2_{1,b}$ and $R^2_{3,b} - R^2_{2,b}$ for the $b = 1, \ldots, 100$ simulations and then collect these differences into a histogram or box plot and show that they never go below 0.*)

Solution : The three models

$$
\begin{aligned}
&\text{Model 1} : Y_i = \beta_0 + \beta_1 X_i + \epsilon_i \\
&\text{Model 2} : Y_i = \beta_0 + \beta_1 X_i + \beta_2 Z_i + \epsilon_i \\
&\text{Model 3} : Y_i = \beta_0 + \beta_1 X_i + \beta_2 Z_i + \beta_3 \sin(X_i) + \epsilon_i
\end{aligned}
$$

Figure 1 shows the box plots of R^2 values for Model 1, Model 2 and Model 3 on training data. It is observed that with more predictors the median R^2 values exhibit a very small increase.

Figure 2 shows the box plots of difference in R^2 values for Model 2 and Model 1, Model 3 and Model 2, and Model 3 and Model 1. This plot reveals that these difference values are always non-negative, indicating that adding complexity never decreases the R^2 value on the training data. Each subsequent model either improves or does not significantly harm the R^2.

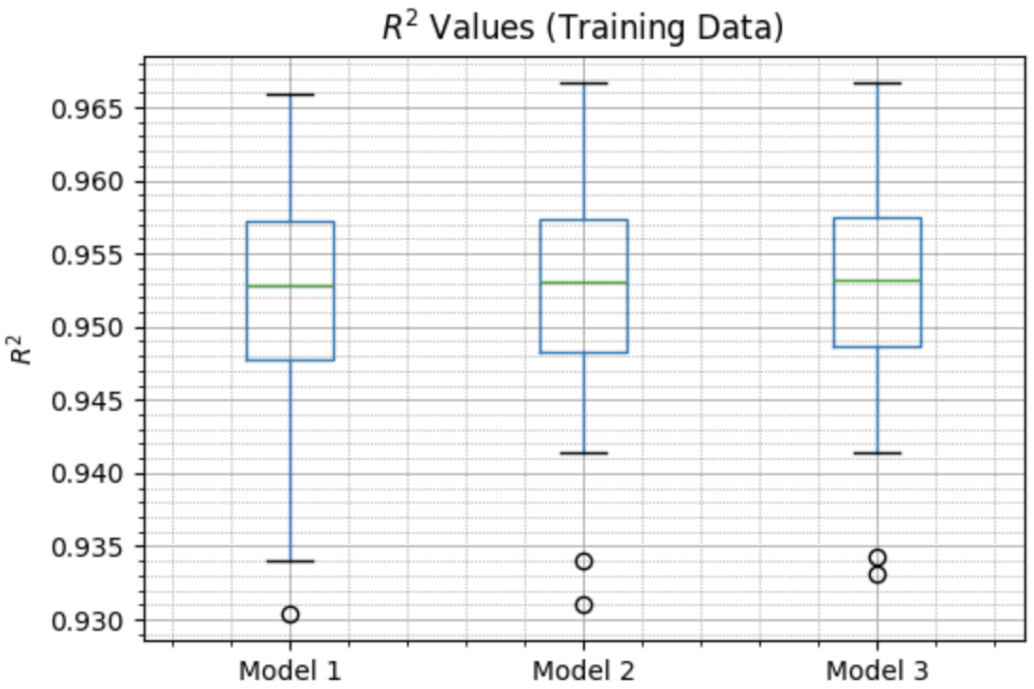


Figure 1: (1d) Box plot of R^2 values for Model 1, 2 and 3 on training set

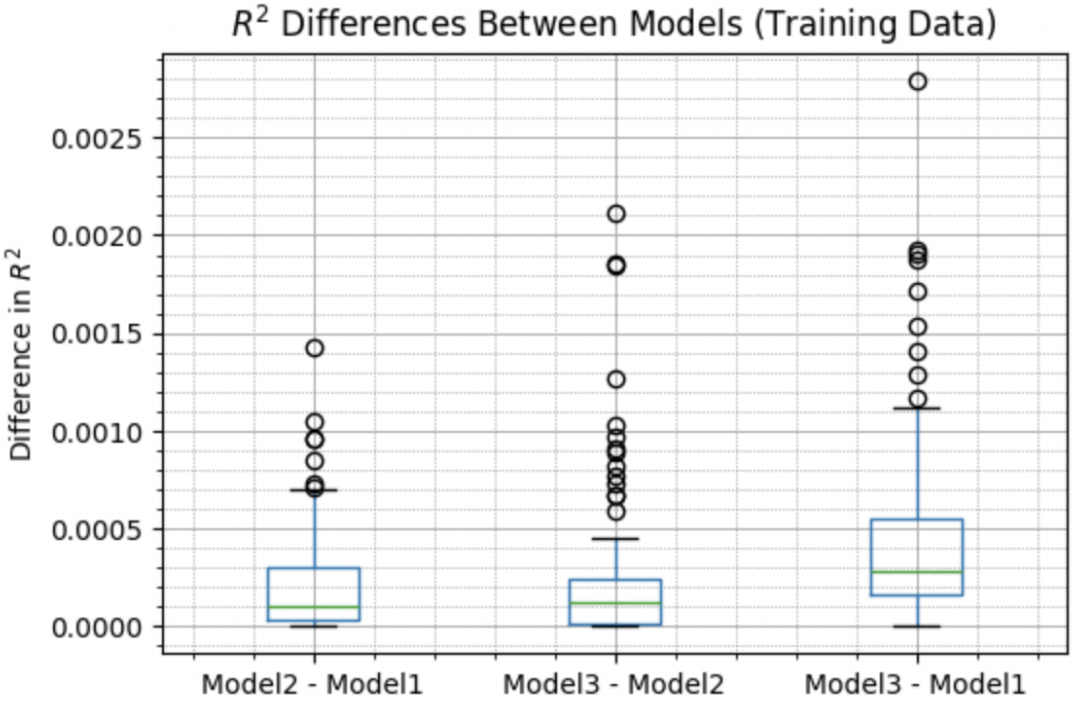


Figure 2: (1d) Box plot of difference in R^2 values between models on training set

(e) In Q1(d), the R^2 that you computed was actually the *training set* R^2, in the sense that the observations you used to compute R^2 are the same as the observations that you used to fit the model $\hat{f}(\cdot)$ and to compute $\bar{y}$ (i.e., you used the training observations).

In this subproblem, compute *test set* R^2 instead. To do this, for each of the 100 simulated training sets in Q1(d), compute a test set consisting of 200 test observations drawn from the same model (1). Then, compute the R^2 over this test set, using $\hat{f}(\cdot)$ and $\bar{y}$ obtained using the *training* observations in the expression for R^2. Display the results.

Solution : For the three models from the part(d), Figure 3 shows the box plots of R^2 values for Model 1, Model 2 and Model 3 on training data. It is observed that with more predictors the median R^2 values are close and they decrease slightly as we go from Model 1 to Model 3.

Figure 4 Shows the box plots of difference in R^2 values for Model 2 and Model 1, Model 3 and Model 2, and Model 3 and Model 1 on test set. The differences are mostly centered near zero or slightly negative when moving to more complex models suggesting potential overfitting.

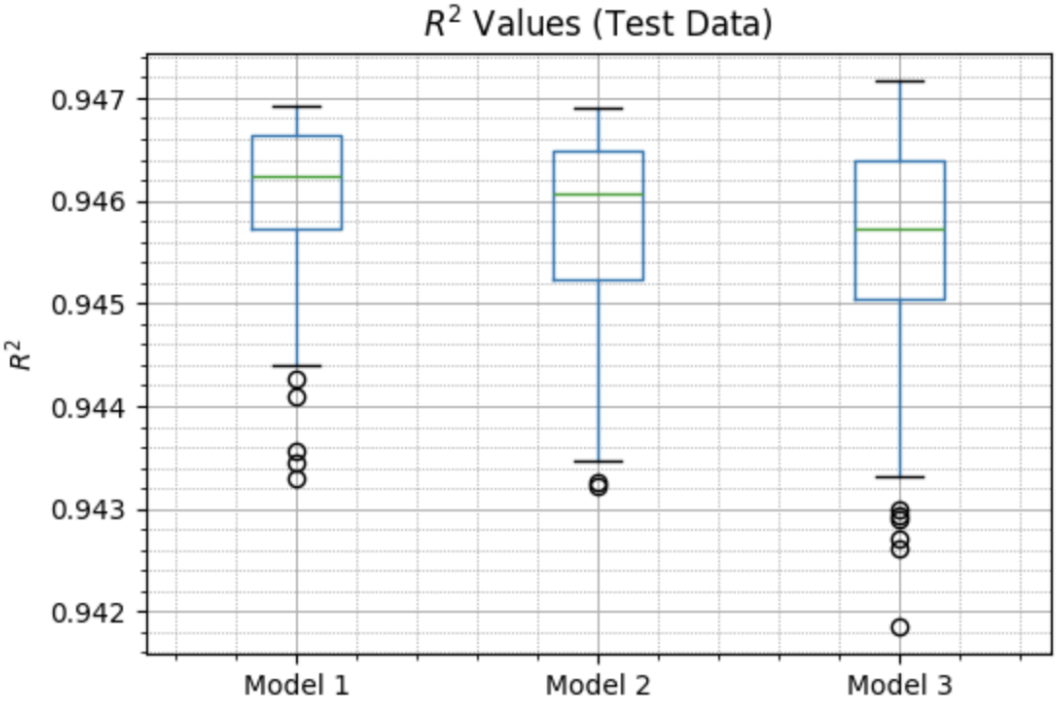


Figure 3: (1e) Box plot of R^2 values for Model 1, 2 and 3 on test set

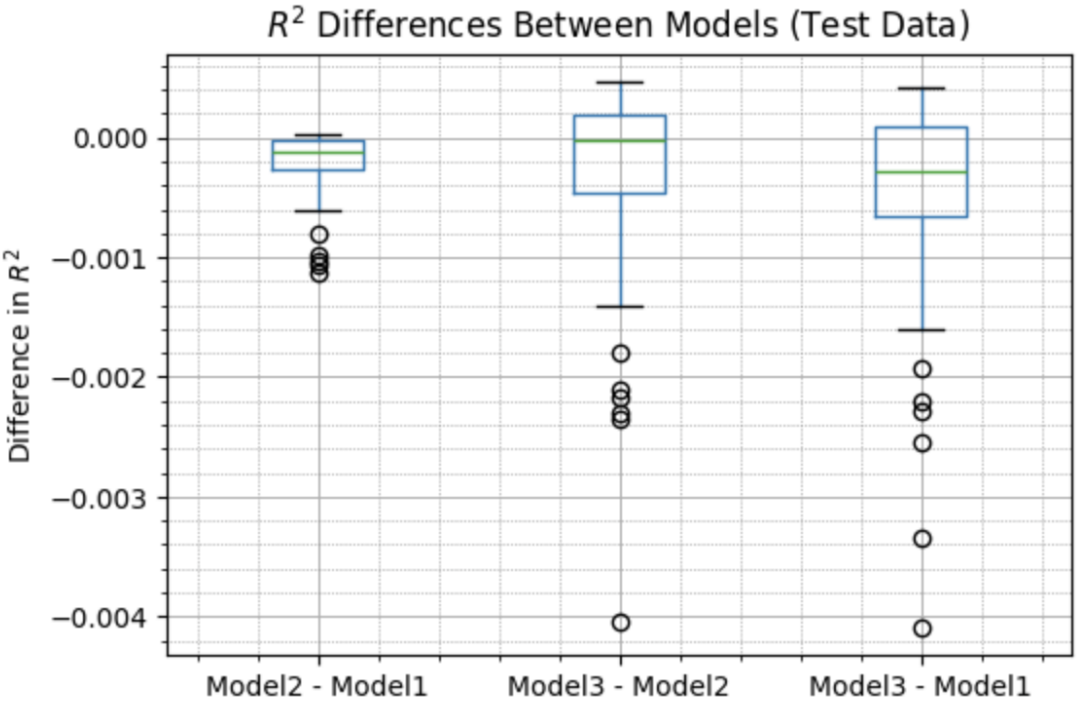


Figure 4: (1e) Box plot of difference in R^2 values between models on test set

(f) How does test set R^2 compare to training set R^2 in each of the three models considered? Explain your answer in terms of the bias-variance trade-off.
Solution : For each of the three models, the test set R^2 is generally lower than the training set R^2 as shown in Figures 1 and 3. This happens because models tend to fit the training data better than unseen data.

Model 1 including only the relevant predictor has relatively higher bias but low variance, allowing it to generalize well to unseen data. Model 2 adds an irrelevant predictor introduces additional variance without reducing bias, potentially leading to overfitting and reduced test performance. Model 3 further increases model complexity by adding a non-linear transformation which may fit noise in the training data increasing variance even more. As a result, although training R^2 increases with model complexity, test R^2 can decrease due to higher variance, demonstrating the classic bias-variance trade-off. Adding complexity can reduce bias but often increases variance.

2. Find a data set (either online, or one of the datasets featured in the textbook) with $p = 2$ features X_1 and X_2, a qualitative response Y with $K = 2$ classes, and at least 20 observations per class. *(If you have a data set with more than two features or more than two classes, then feel free to just select a subset of the features and classes so that you can use the data for this problem.)* We are

going to predict Y using X_1 and X_2.

First, divide your data into a training set and a test set by reserving 75% of your data for training and 25% for testing. Problems (a) through (e) below will involve only your training data, and then you will bring in your test data in part (g).

(a) Briefly describe the data. Where did you get it? Describe the $K = 2$ classes and the $p = 2$ features. Explain the classification task in words (e.g. a sentence along the lines of "I will use the expression levels of genes ABC and DEF to predict whether a patient belongs to class G or H.")
Solution : Dataset Used: S&P Stock Market Data
Source: Raw values of the S&P 500 were obtained from Yahoo Finance and then converted to percentages and lagged.
The dataset consists of the daily percentage returns for the S&P 500 stock index from 2001 to 2005. I aim to use the returns from the past two days (Lag1 and Lag2) to predict whether the market will go up or down on the following day. The classification task uses two features:

- Lag1 (X_1): Percentage return for the previous trading day
- Lag2 (X_2): Percentage return for two days prior

The response variable Direction(Y) indicates whether the market had a positive ("Up") [$Y = 1$] or negative ("Down") [$Y = 0$] return on a given day.

(b) Fit an LDA model to the data. Make a plot with X_1 and X_2 on the horizontal and vertical axes, and with the observations displayed and colored according to their true class labels. On the plot, indicate which observations are incorrectly classified.
Solution : Figure 5 shows the LDA classification on the training data.
Training error: 45.99%

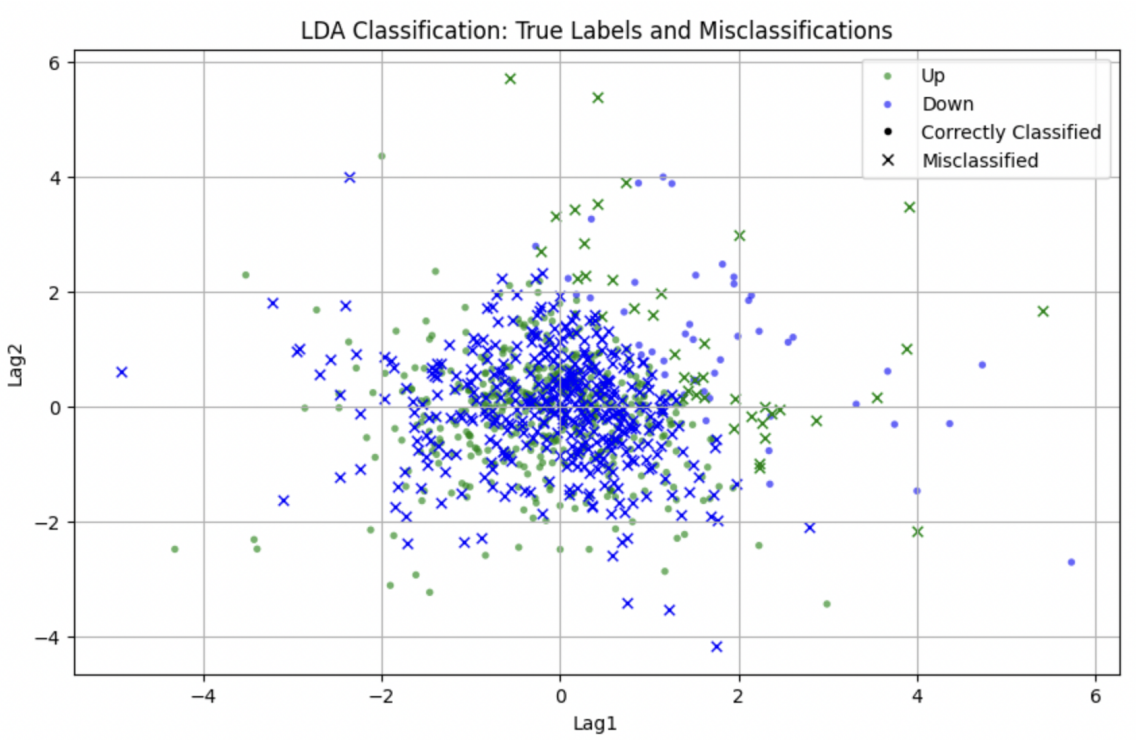


Figure 5: (2b) LDA Classification on Training Data

(c) Fit a k-nearest neighbors classifier to the data using $k = 6$. Make a plot with X_1 and X_2 on the horizontal and vertical axes, and with the observations displayed and colored according to their true class labels. On the plot, indicate which observations are incorrectly classified.
Solution : Figure 6 shows the k-nearest neighbors classification using $k = 6$ on the training data.
Training error: 35.43%

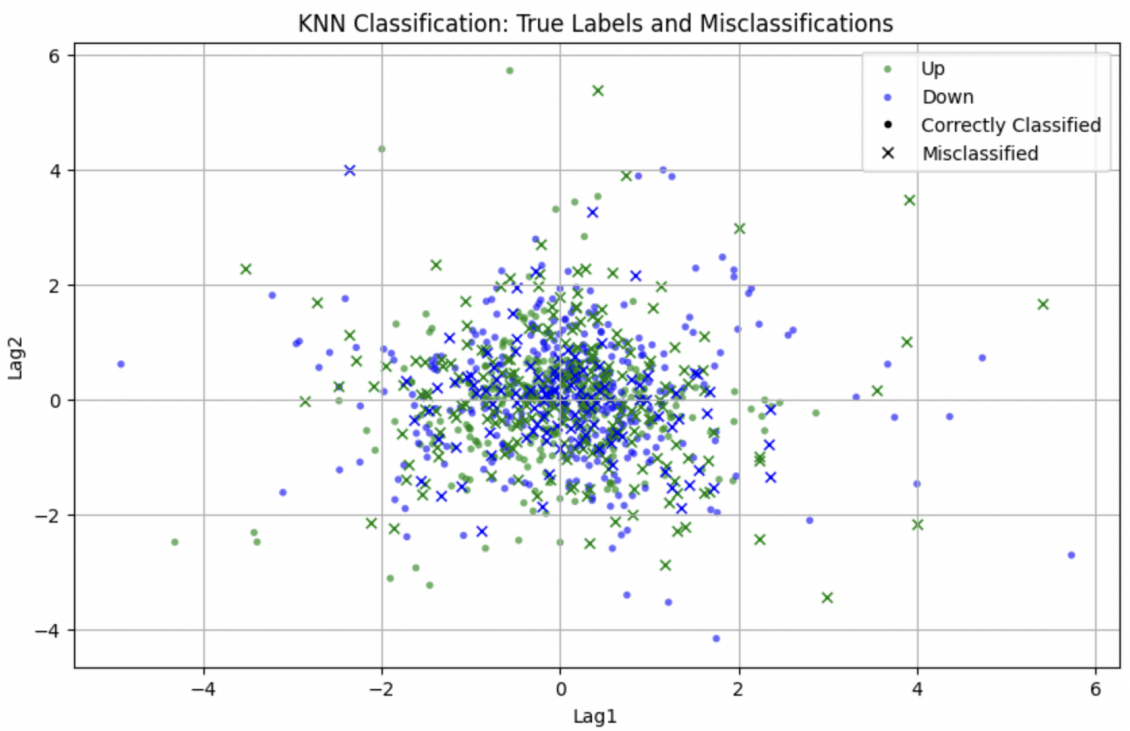


Figure 6: (2c) KNN(k=6) Classification on Training Data

(d) Fit a logistic regression model to the data. Make a plot with X_1 and X_2 on the horizontal and vertical axes, and with the observations displayed and colored according to their true class labels. On the plot, indicate which observations are incorrectly classified.
Solution : Figure 7 shows the logistic regression model on the training data.
Training error: 46.10%

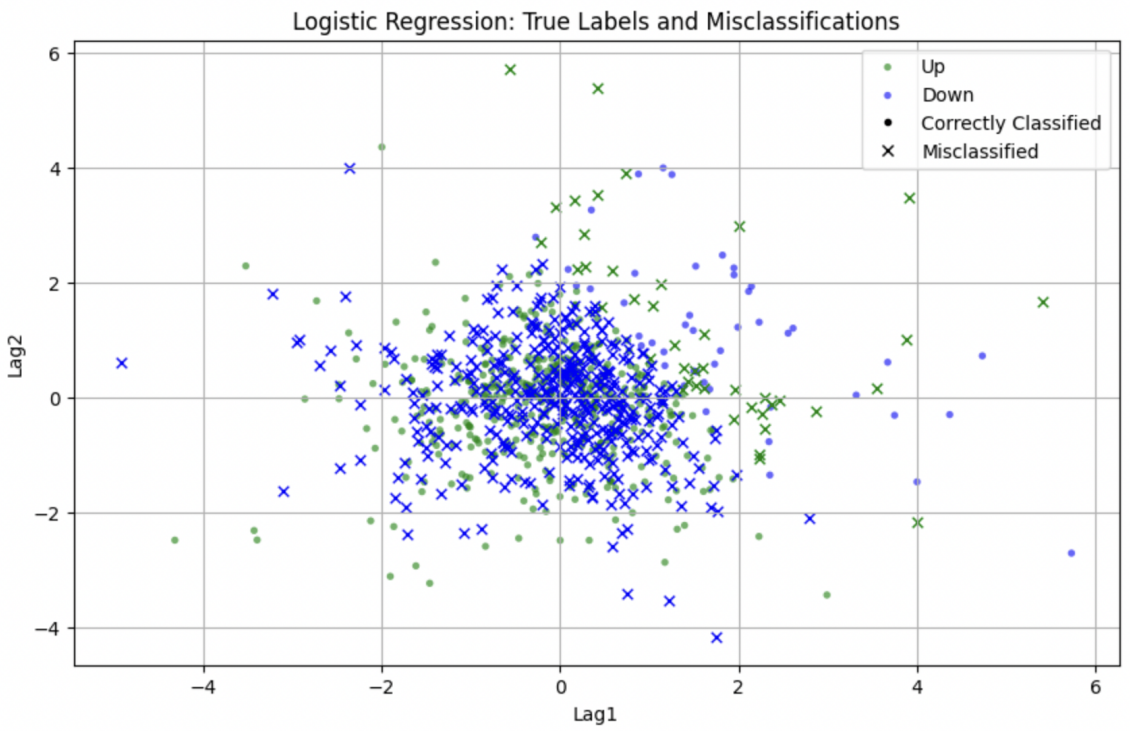


Figure 7: (2d) Logistic Regression Predictions on the Training data

(e) Out of the three models, which one gave you the smallest training error?
Solution : Training error rates for the three models:

- LDA = 45.99%
- KNN (k=6) = 35.43%
- Logistic Regression = 46.10%

Out of the three models, KNN (k=6) model gave the smallest training error of 35.43%.

(f) *Before making a formal evaluation using your test set,* which of the three models above do you intuitively suspect will do the best on your test set? Why do you think so? (Leave your guess as-is in your response to this problem before moving on to part (g).)
Solution : Since KNN had the smallest training error and the data does not have a clear linear decision boundary, I expect KNN to perform the best out of the three classifiers (LDA, KNN, and logistic regression) on the test data as the other two are linear classifiers.

(g) Using your test set, evaluate the performance for each model. Because we have $K = 2$ classes, choose one of your classes to be "positive" and one of your classes to be "negative." (In some cases, this choice will be very easy as your classes may correspond to some notion of positivity and negativity. For other types of data sets, this choice may be totally arbitrary.)

For each model, report each of the following accuracy metrics:

- Proportion of correct predictions on the test set
- False negative rate (the proportion of times that your model predicts a "negative" class out of all the instances where the test point was actually "positive.")
- False positive rate (the proportion of times that your model predicts a "positive" class out of all the instances where the test point was actually "negative.")

Solution : We have two classes: "Up" and "Down". We define "Up" as the positive class and "Down" as the negative class. Using the test set accuracy merics for the three models:

i. LDA Model
   - Proportion of correct predictions on the test set = 0.476
   - False negative rate = 0.0952
   - False positive rate = 0.9036

ii. KNN Model with $k = 6$
   - Proportion of correct predictions on the test set = 0.4792
   - False negative rate = 0.5850
   - False positive rate = 0.4639

iii. Logistic Regression Model
   - Proportion of correct predictions on the test set = 0.4792
   - False negative rate = 0.0952
   - False positive rate = 0.8976

(h) Is there a clear winner from your result in 2(g)? **Based upon the context of your chosen data set,** do you think that it would be *more* important to minimize the false negative rate, the false positive rate, or some combination of the two? Explain your answer.
Solution : All models turn out to have very similar test accuracy (around 47%–48%). There is no clear winner among the models just based on the accuracy. However, when we examine the false negative rates (FNR) and false positive rates (FPR), we notice important differences. Both the LDA model and the logistic regression model achieve very low false negative rates (9.5%) but suffer from extremely high false positive rates (90%). Meanwhile, the KNN model has a much higher false negative rate (58.5%) but a significantly lower false positive rate (46.4%).

For this problem, predicting whether the stock market will go up or down, minimizing the false positive rate is more critical. A false positive in this setting means incorrectly predicting that the market will rise when it will actually fall. This could lead to investment decisions that result in financial losses. In contrast, a false negative (predicting a market drop when it actually rises) would mostly mean missed opportunities rather than direct financial loss.

Although the KNN model has a higher false negative rate, its considerably lower false positive rate makes it more suitable for this problem. Based on the context, minimizing false positives is more important, hence the KNN model would likely be the better choice among the three.

3. Suppose we have a binary response $Y \in \{0, 1\}$, and a single predictor $X \in \mathbb{R}$. This question does not make any sort of modeling assumption (e.g. there is no need to assume that $\Pr(Y = 1 \mid X = x) = \frac{\exp(\beta_0+\beta_1 X)}{(1+\exp(\beta_0+\beta_1 X))}$). Recall that the odds is defined as $\Pr(Y = 1 \mid X = x)/\Pr(Y = 0 \mid X = x)$.

   (a) Suppose that for a given value x, the log-odds equals 0.7. What is $\Pr(Y = 1 \mid X = x)$? What is $\Pr(Y = 0 \mid X = x)$? (Your answer should be an actual number, like 0.2323 but not that number specifically.)
   Solution : log-odds = 0.7

$$\log(\Pr(Y = 1 \mid X = x)/\Pr(Y = 0 \mid X = x)) = 0.7$$

$$\Pr(Y = 1 \mid X = x)/\Pr(Y = 0 \mid X = x) = e^{0.7} \approx 2.0138$$

$$\text{Let } Pr(Y = 1 \mid X = x) = p$$

$$\therefore Pr(Y = 0 \mid X = x) = 1 - p$$

$$\Pr(Y = 1 \mid X = x)/\Pr(Y = 0 \mid X = x) = p/(1 - p) \approx 2.0138$$

$$p = 2.0138(1 - p)$$

$$p = 2.0138/3.0138 \approx 0.6682$$

$$1 - p \approx 0.3318$$

$$\mathbf{Pr(Y = 1 \mid X = x) \approx 0.6682}$$

$$\mathbf{Pr(Y = 0 \mid X = x) \approx 0.3318}$$

   (b) Now for another value x_0, suppose that $\Pr(Y = 1 \mid X = x_0) = 0.3$. What is the log-odds? (Your answer should be an actual number, like -89.23211 but not that number specifically.)
   Solution :

$$\Pr(Y = 1 \mid X = x_0) = 0.3$$

$$\Pr(Y = 0 \mid X = x_0) = 1 - 0.3 = 0.7$$

$$\text{Log-odds} = \log(\Pr(Y = 1 \mid X = x_0)/\Pr(Y = 0 \mid X = x_0))$$

$$\text{Log-odds} = \log(0.3/0.7) = \mathbf{-0.8473}$$

4. Suppose we have a binary response $Y \in \{0, 1\}$, and a single predictor $X \in \mathbb{R}$, and that $\Pr(Y = 1 \mid X = x) = \exp(\beta_0 + \beta_1 x)/(1 + \exp(\beta_0 + \beta_1 x))$. A little birdie tells you that for $x = 1$, we have $P(Y = 1 \mid X = 1) = 0.9$ and $P(Y = 1 \mid X = 3) = 0.5$.

(a) What are the values of β_0 and β_1? (Your answer should be an actual number, like $\beta_0 = 0.232$ and $\beta_1 = -232444$, but not that number specifically.)
Solution : Given,

$$\Pr(Y = 1 \mid X = 1) = \exp(\beta_0 + \beta_1(1))/(1 + \exp(\beta_0 + \beta_1(1))) = 0.5$$

$$\text{Let, } \exp(\beta_0 + \beta_1(1) = t$$

$$\therefore \Pr(Y = 1 \mid X = 1) = t/(1 + t) = 0.9$$

$$\text{Solving for t gives : } t = 9$$

$$\implies \exp(\beta_0 + \beta_1) = 9$$

$$\implies \beta_0 + \beta_1 = ln(9) \text{ (Equation 1)}$$

Given,

$$\Pr(Y = 1 \mid X = 3) = \exp(\beta_0 + \beta_1(3))/(1 + \exp(\beta_0 + \beta_1(3))) = 0.5$$

$$\text{Let, } \exp(\beta_0 + \beta_1(3)) = w$$

$$\therefore \Pr(Y = 1 \mid X = 3) = w/(1 + w) = 0.5$$

$$\text{Solving for w gives : } w = 1$$

$$\implies \exp(\beta_0 + \beta_1(3)) = 1$$

$$\implies \beta_0 + \beta_1(3) = ln(1) = 0 \text{ (Equation 2)}$$

Using the two equations Equation(2)-Equation(1), we get

$$2 * \beta_1 = -ln(9)$$

$$\beta_1 = -ln(9)/2 = -1.0986$$

Using Equation 1 and Equation 2 we now find β_0 by using β_1

$$\text{Equation 1 : } \beta_0 + \beta_1 = ln(9)$$

$$\implies \beta_0 = ln(9) + ln(9)/2$$

$$\implies \beta_0 = 3 * ln(9)/2 = 3.2958$$

$$\boldsymbol{\beta_0 = 3.2958} \text{ and } \boldsymbol{\beta_1 = -1.0986}$$

(b) Make a plot with X on the horizontal axis and Y on the vertical axis. Using the values of β_0 and β_1 that you calculated in Q4(a), display $\Pr(Y = 1 \mid X = x)$ with a solid line and $\Pr(Y = 0 \mid X = x)$ using a dashed line.
Solution : Figure 8 shows $\Pr(Y = 1 \mid X = x)$ and $\Pr(Y = 0 \mid X = x)$ calculated using the values of β_0 and β_1 from part(a).

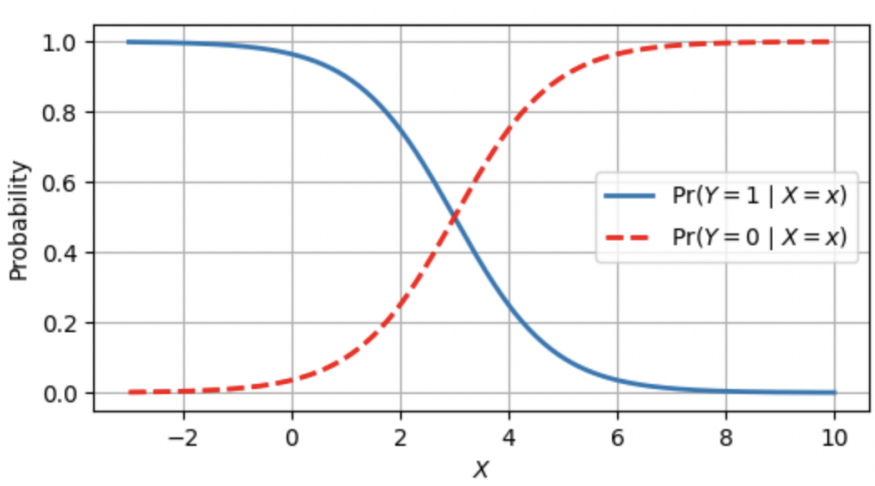


Figure 8: (4b) Probability v/s X ($logit(\text{p}) = \beta_0 + \beta_1 X$)

5. We will now see that replacing $\beta_0 + \beta_1 X$ with a more flexible function in the expression for logistic regression leads to a more flexible shape for $\Pr(Y = 1 \mid X = x)$.

   Make a plot with X on the horizontal axis and Y on the vertical axis. With $\beta_0 = 0.2$, $\beta_1 = -0.7$, and $\beta_2 = 0.6$, display

   $$\Pr(Y = 1 \mid X = x) = \frac{\exp(\beta_0 + \beta_1 x + \beta_2 x^2)}{1 + \exp(\beta_0 + \beta_1 x + \beta_2 x^2)}.$$

   Make sure to choose a range for the horizontal axis that allows you to see the full shape of the function.

   Comment on the shape of this function. How does it compare to the shape of the logistic function in Q4(b)? Why did replacing $\beta_0 + \beta_1 X$ with $\beta_0 + \beta_1 X + \beta_2 X^2$ have this effect?

   Solution : Figure 9 shows $\Pr(Y = 1 \mid X = x)$ and $\Pr(Y = 0 \mid X = x)$ calculated using the values $\beta_0 = 0.2$, $\beta_1 = -0.7$, and $\beta_2 = 0.6$.

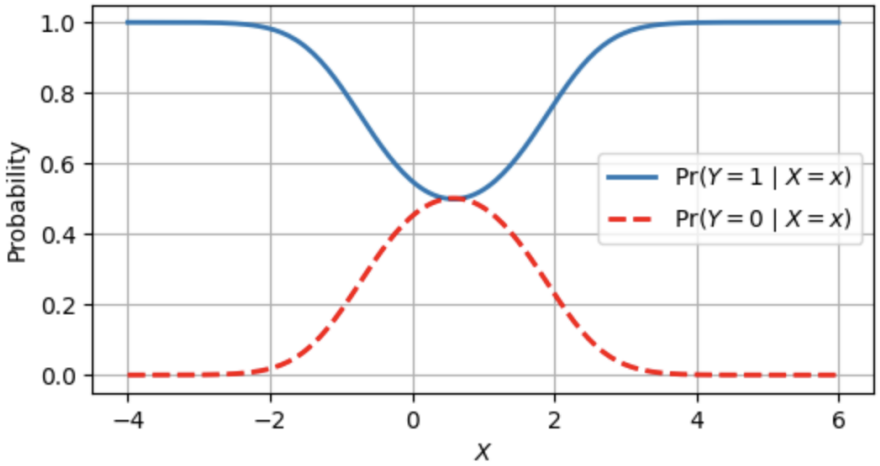


Figure 9: (5) Probability v/s X ($logit(\text{p}) = \beta_0 + \beta_1 X + \beta_2 X^2$)

The probability function is parabolic (non-monotic). The probability $Pr(Y = 1|X = x)$ dips in the middle and rises again on both ends while $Pr(Y = 0|X = x)$ forms a peak in the middle. This happens because the model includes a quadratic term X^2. As X moves away from 0 the quadratic X^2 term becomes dominant, causing the exponent of the logistic function to increase. This results in probabilities bending and curving rather than smoothly transitioning. In Question 4, the shape of the logistic function is standard "S"-shaped sigmoid (monotonic) as X increases the probability $Pr(Y = 1|X = x)$ decreases smoothly from 0 to 1 while $Pr(Y = 0|X = x)$ increases smoothly.

6. In this problem, you will simulate some classification data with 3 classes. Your simulated data should have $p = 2$ features (so that you can easily plot it), a training set of 200 observations, and a test set of $2,000$ observations.

   You will compare the test error of two different classifiers: a K-nearest neighbors (KNN) classifier with 5 neighbors, and a classifier that uses a linear decision boundary (you can decide whether to use LDA for this, or multinomial logistic regression — I think the former is slightly easier in `R` and the latter is slightly easier in `Python`. The choice is yours; please just please clarify in your HW which one you used).

   The goal of this problem is to figure out how to generate the data in two different ways: so that in (a) the KNN classifier will have lower test error, and in (b) the linear decision boundary classifier will have lower test error.

   (a) First, simulate the data in such a way that KNN classifier with 5 nearest neighbors has lower test error than the linear decision boundary classifier.

      i. Describe the simulation setting: how did you generate the data, and why did this lead to KNN having a lower test error than the linear decision boundary classifier?
      Solution : I will be using KNN (k=5) and LDA classifiers to model the data. In this simulation, two-dimensional data points arranged in concentric circular regions (rings) were generated corresponding to three different classes:

      - Class 0: Uniformly sampled in a circle of radius 2, centered at 0.
      - Class 1: Uniformly sampled within ring of radius between 2 and 5, centered at 0.
      - Class 2: Uniformly sampled within a ring of radius between 5 and 7, centered at 0.

      The two features X_1 and X_2 were the polar coordinates:

      $$X_1 = rcos(\theta) \text{ and } X_2 = rsin(\theta)$$

      This setup was used to generate both the training set (200 points: roughly 66–67 points per class) and the test set (2000 points: 666–667 points per class).Figure 10 shows the simulated training data.

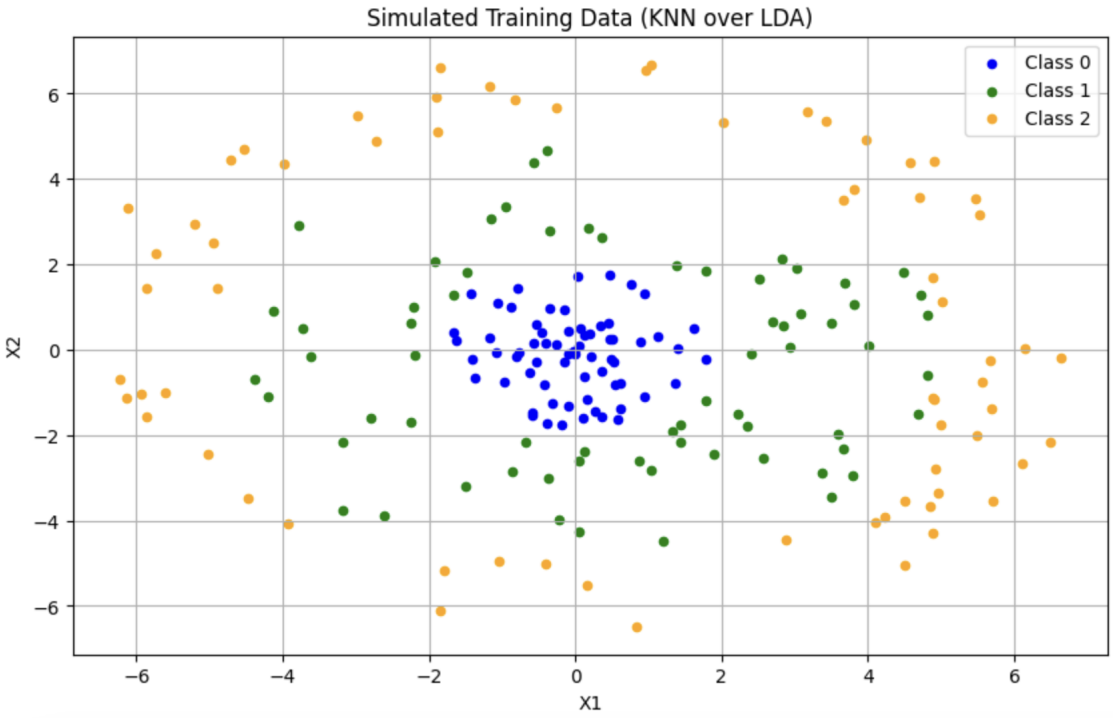


Figure 10: (6a) Simulated Training Data where KNN will perform better than LDA

The three classes are separated by circular boundaries and not straight lines or planes. Since LDA assumes linear decision boundaries, it struggles to correctly classify points in this setting. On the other hand, KNN is non parametric method and classifies based on nearest training labels. It can model nonlinear decision boundaries, hence KNN performs better than LDA in this setting.
Model accuracies (test data): KNN(k=5) = 0.939 ; LDA = 0.3895
Error Rates (test data): KNN(k=5) = 6.1% ; LDA = 61.05%

ii. Make a plot that displays the *training* observations, as well as the decision boundary corresponding to KNN. Make another plot that displays the *test* observations, as well as the decision boundary from KNN that you obtained from the training data. Then, make these two plots again, but this time displaying the decision boundary corresponding to the linear decision boundary classifier. Within each plot, find a way to indicate which training and test observations are mislabeled by the corresponding classifier.
Note: The horizontal and vertical axes of your plots should be X_1 and X_2. You might want to use different colors to represent the three classes. You might want to use different symbol types to represent the four types of observations: correctly versus incorrectly classified training observations, and correctly versus incorrectly classified test observations. Be sure to include a legend, and to fully label all aspects of your plot so that it is understandable!

Solution : In all the plots, Class 0 is represented with blue, Class 1 with green and Class 2 with orange. The X markers show the misclassified points.
Figure 11 displays the training observations and the decision boundary corresponding to KNN. Figure 12 displays the test observations and the decision boundary corresponding to training KNN. Figure 13 displays the training observations and the decision boundary corresponding to LDA. Figure 14 displays the test observations and the decision boundary corresponding to training LDA.

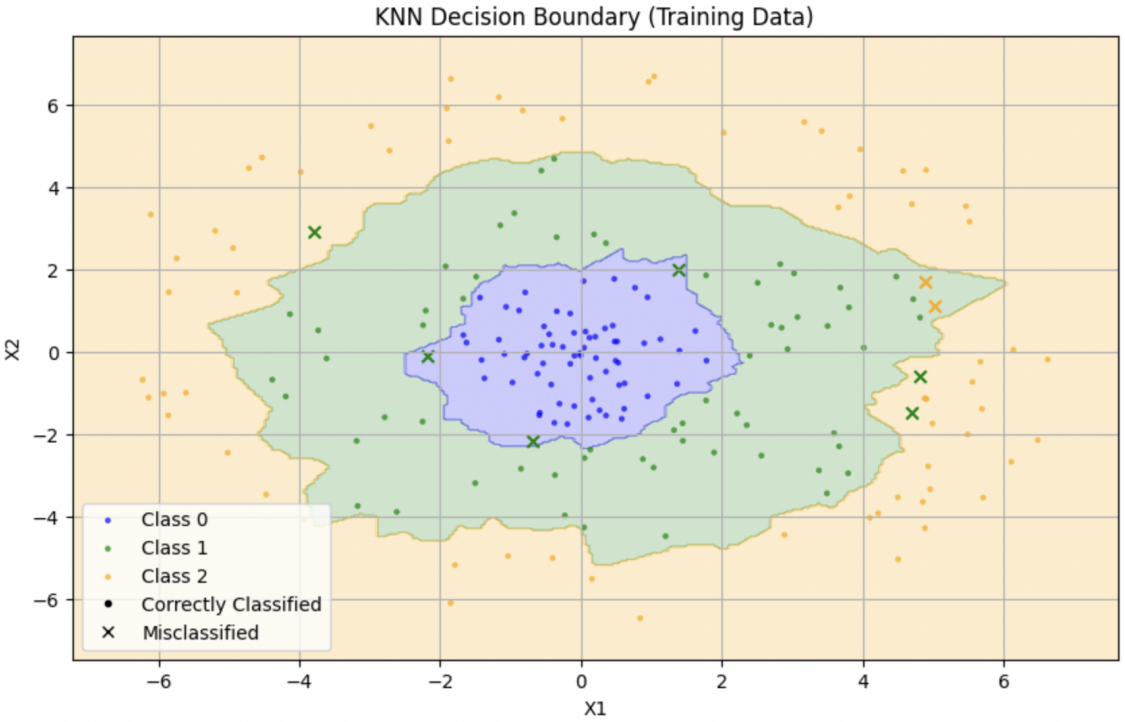


Figure 11: (6a(ii)) KNN(k=5) Decision Boundary (Training Data)

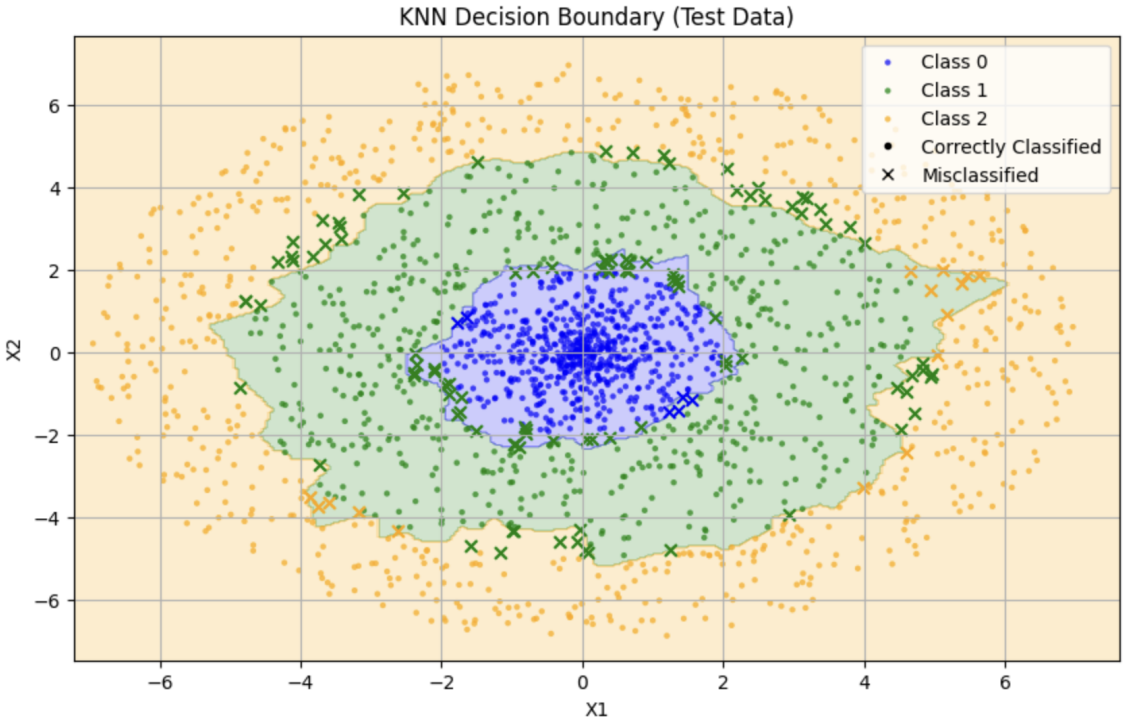


Figure 12: (6a(ii)) KNN(k=5) Decision Boundary (Test Data)

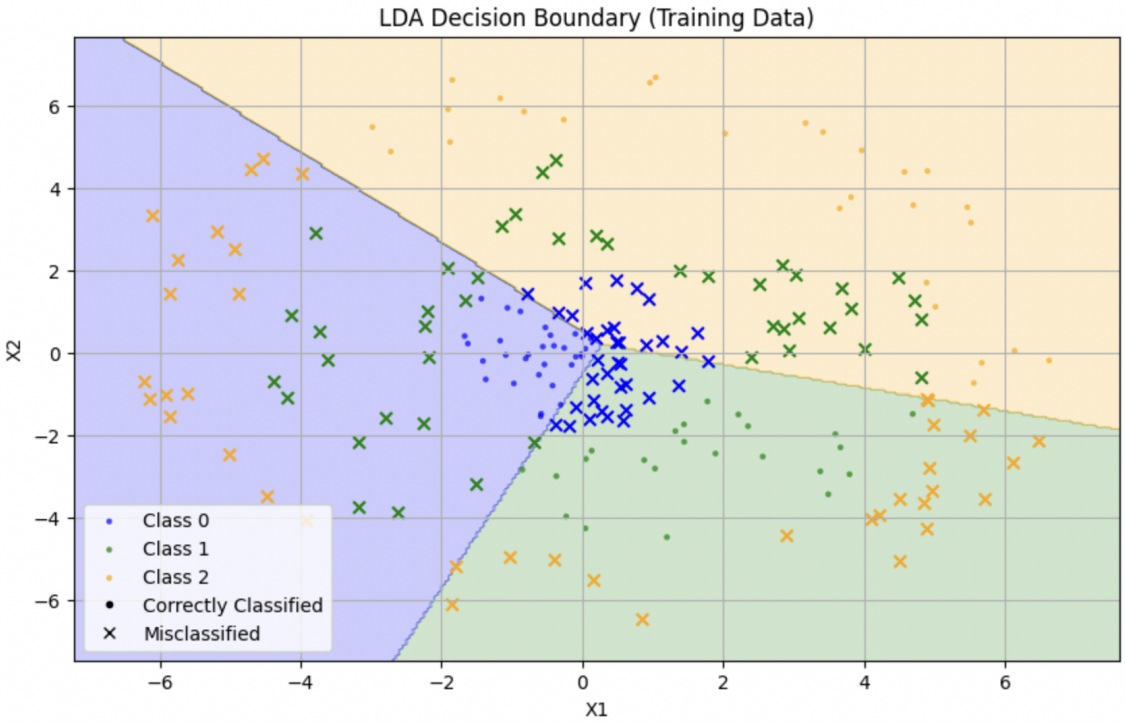


Figure 13: (6a(ii)) LDA Decision Boundary (Training Data)

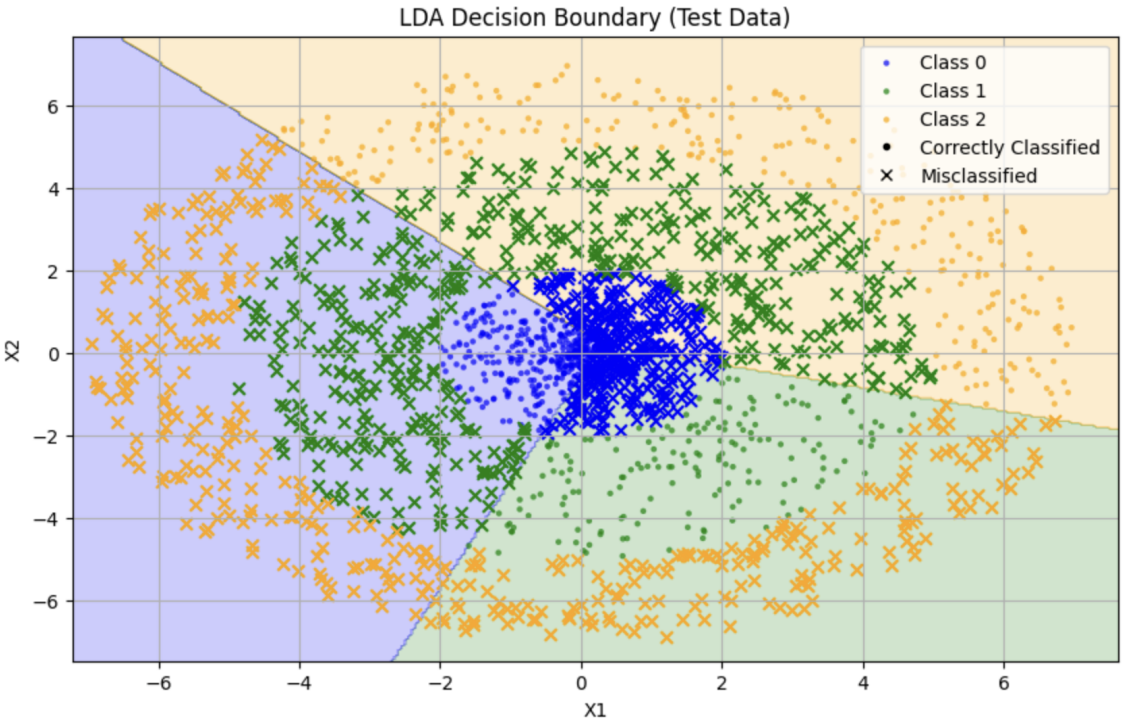


Figure 14: (6a(ii)) LDA Decision Boundary (Test Data)

(b) Repeat Q6(a), but in a simulation setting for which the linear decision boundary classifier has lower test error than KNN.
Solution : I will be using KNN (k=5) and LDA classifiers to model the data. In this simulation, two-dimensional data points were generated corresponding to three different classes. The data for each class was sampled from a bivariate normal distribution with independent features and standard deviation of 1 for both X_1 and X_2 with different means:

- Class 0: sampled from a normal distribution centered at (0,0).
- Class 1: sampled from a normal distribution centered at (1,1).
- Class 2: sampled from a normal distribution centered at (0,3).

This setup was used to generate both the training set (200 points: roughly 66–67 points per class) and the test set (2000 points: 666–667 points per class). Figure 15 shows the simulated training data.

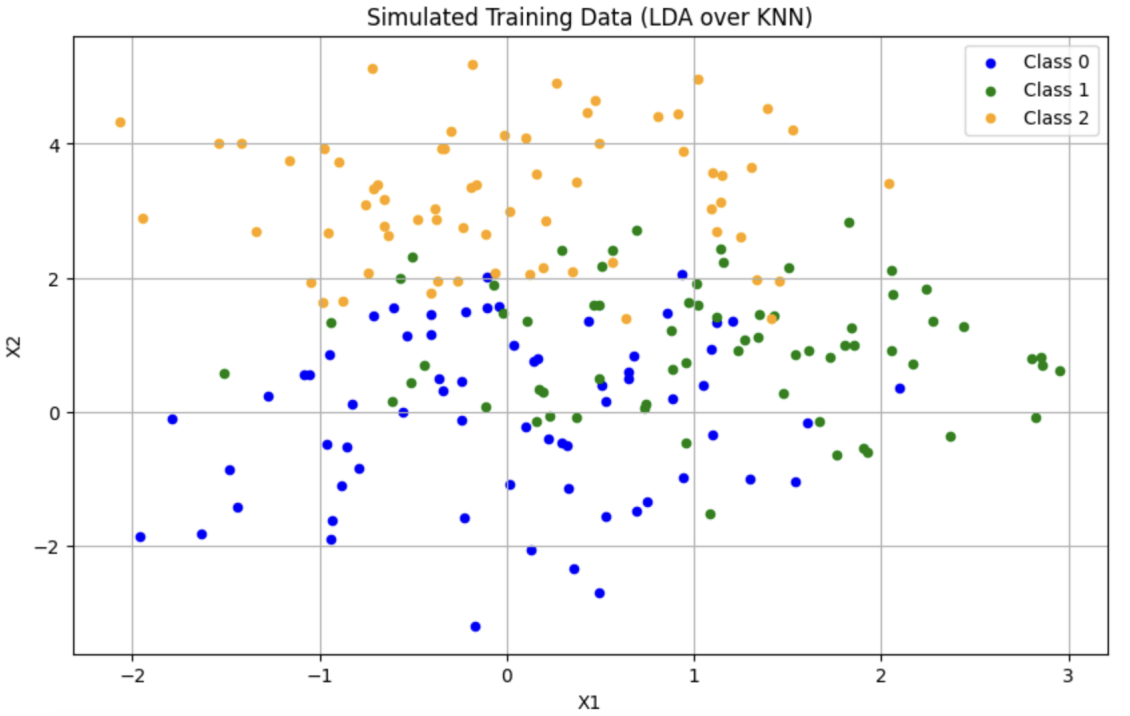


Figure 15: (6b) Simulated Training Data where LDA will perform better than KNN

The three classes are approximately linearly separable because the Gaussians are centered at different means and the separation occurs roughly along straight lines. However, the class means are not very far apart there is some overlap between the distributions. As a result a linear classifier like LDA is likely to perform better than KNN, which can be more sensitive to local variations and noise.
Model accuracies on test data: KNN(k=5) = 0.714 ; LDA = 0.755
Error rates on test data: KNN(k=5) = 28.6% ; LDA = 24.45%

In all the plots, Class 0 is represented with blue, Class 1 with green and Class 2 with orange. The X markers show the misclassified points.
Figure 16 displays the training observations and the decision boundary corresponding to KNN. Figure 17 displays the test observations and the decision boundary corresponding to training KNN. Figure 18 displays the training observations and the decision boundary corresponding to LDA. Figure 19 displays the test observations and the decision boundary corresponding to training LDA.

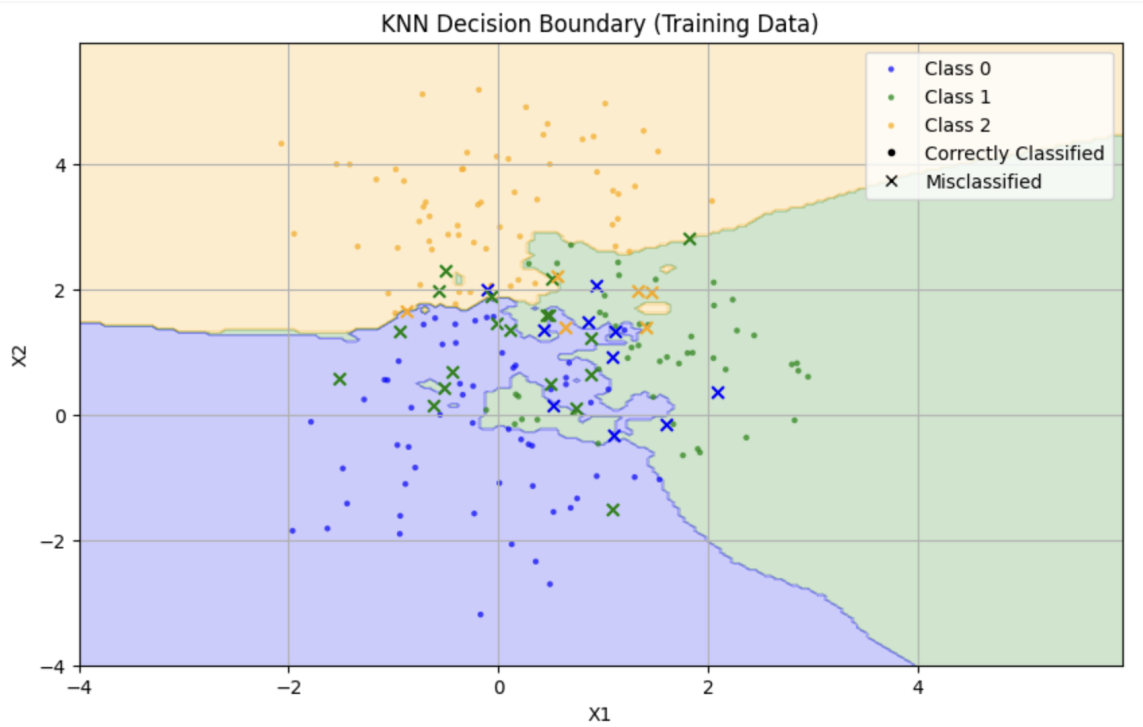


Figure 16: (6b) KNN(k=5) Decision Boundary (Training Data)

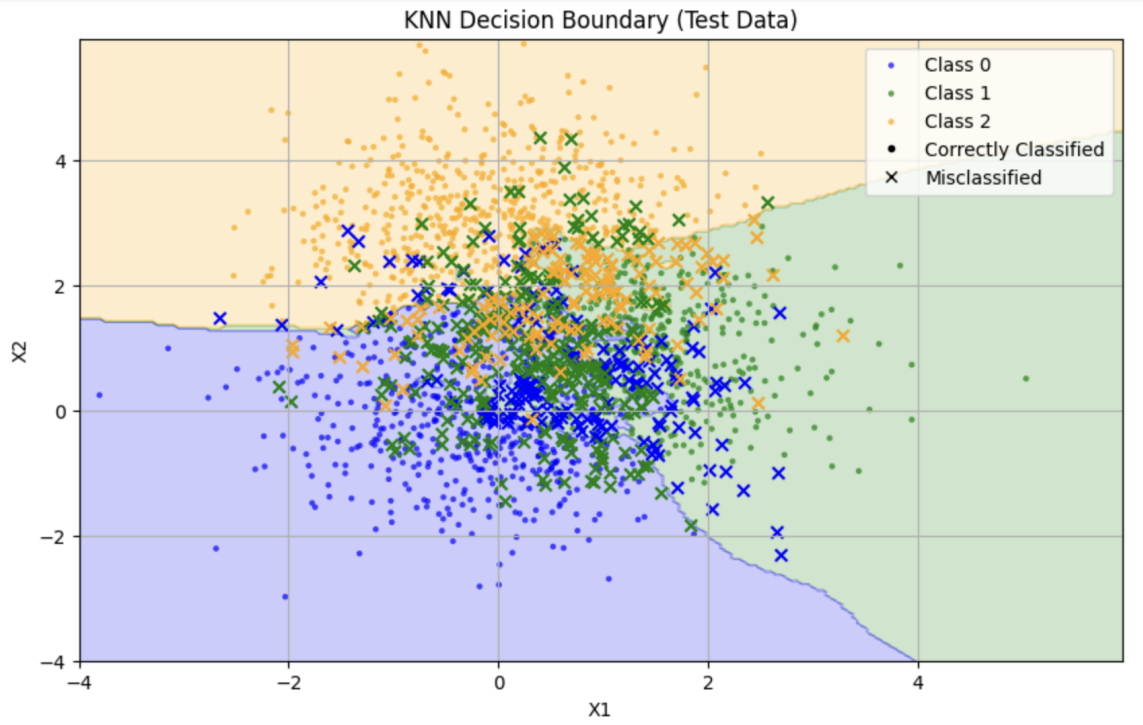


Figure 17: (6b) KNN(k=5) Decision Boundary (Test Data)

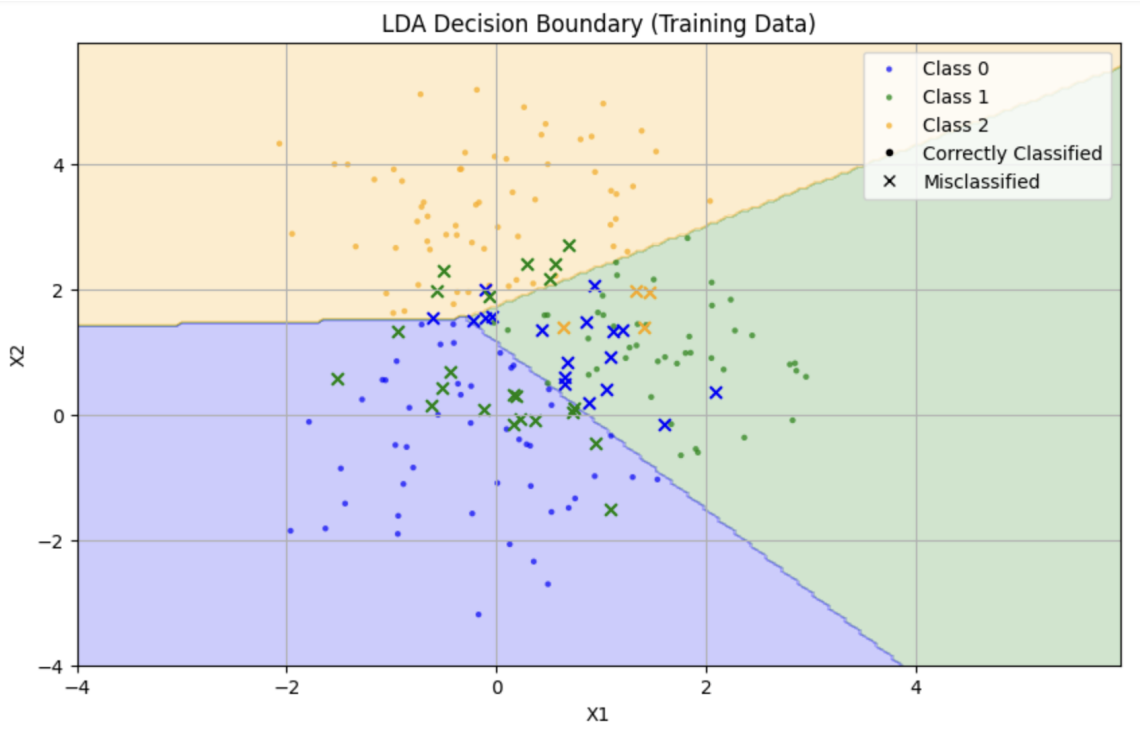


Figure 18: (6b) LDA Decision Boundary (Training Data)

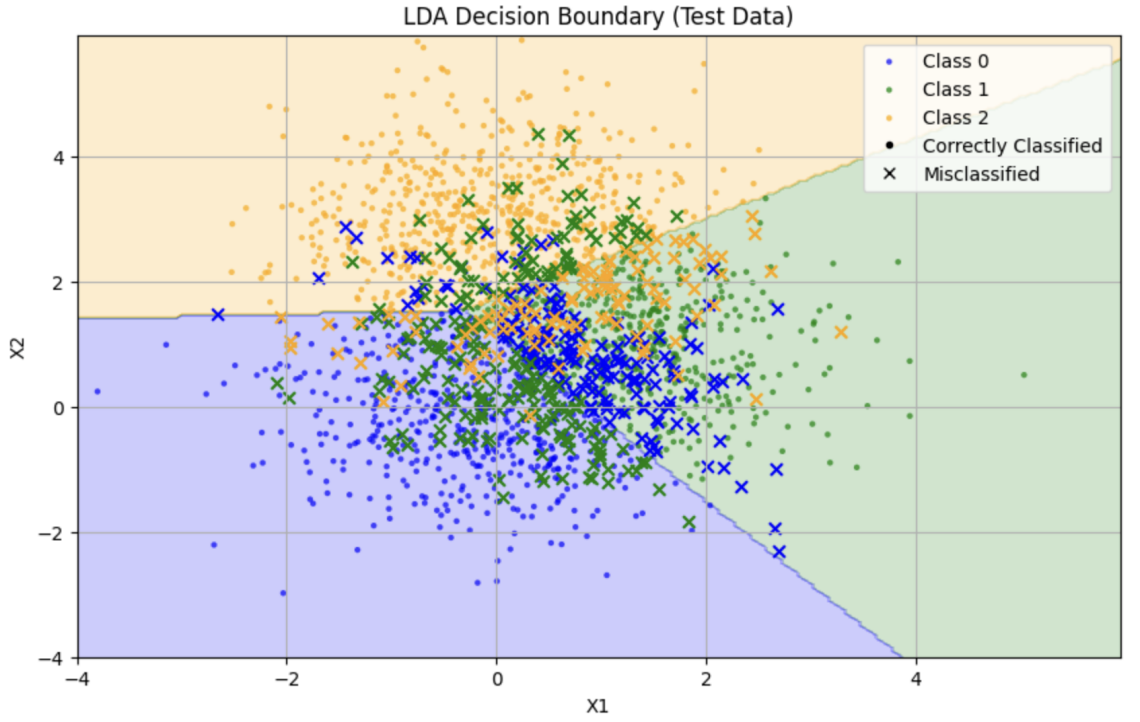


Figure 19: (6b) LDA Decision Boundary (Test Data)

558_HW2

April 30, 2025

[1]:
```
###       APPENDIX     ###
```

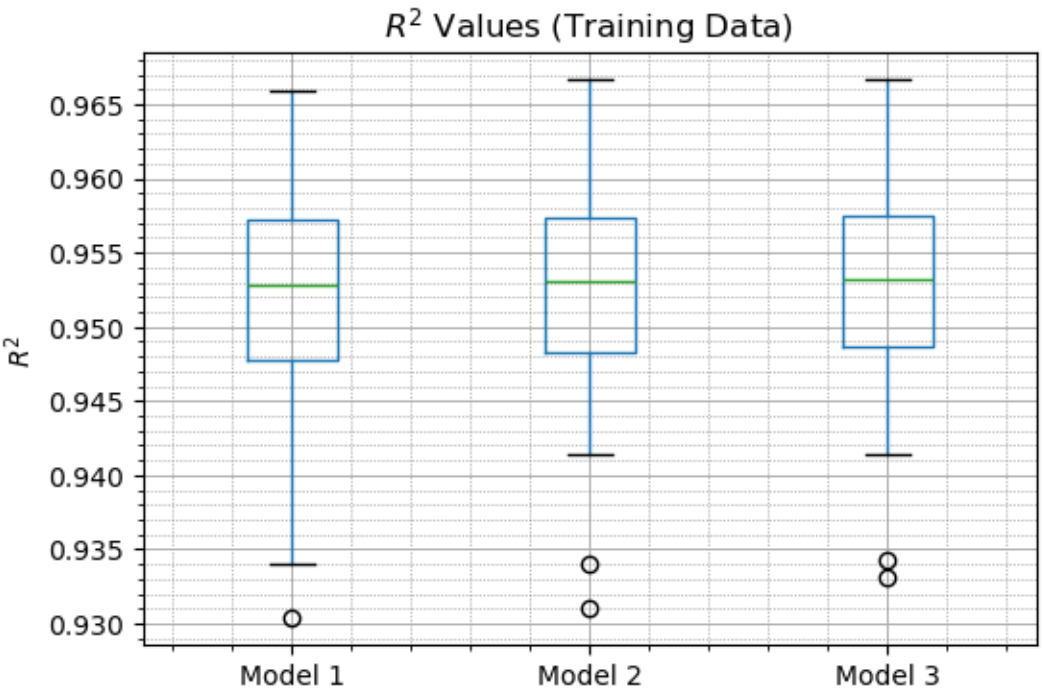
[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
from matplotlib.colors import ListedColormap
```

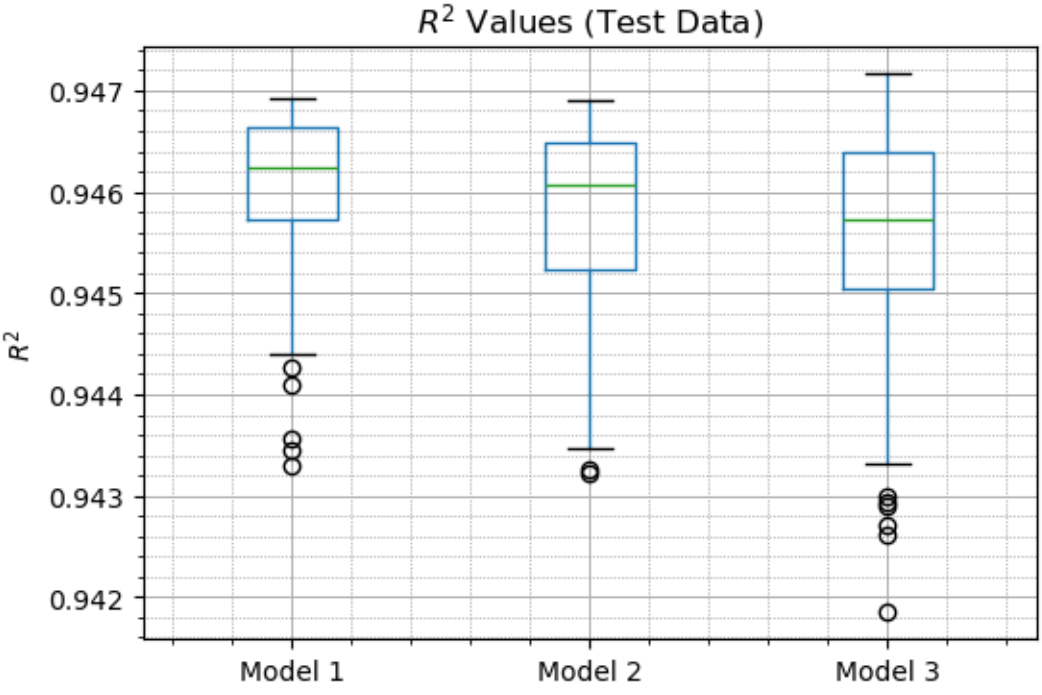
[3]:
```
## Question 1
```

[4]:
```
# Question 1d
np.random.seed(42)
n=200
no_of_sim = 100
r2_values = []
r2_values_test = []
X_test = np.random.normal(0, 3, n)
Z_test = np.random.exponential(scale=4, size=n)
epsilon_test = np.random.normal(0, 2, n)
Y_test = 2 - 3*X_test + epsilon_test
X1_test = X_test.reshape(-1, 1)
X2_test = np.column_stack((X_test, Z_test))
X3_test = np.column_stack((X_test, Z_test, np.sin(X_test)))

for _ in range(no_of_sim):
  X = np.random.normal(0, 3, n)
  Z = np.random.exponential(scale=4, size=n)
  epsilon = np.random.normal(0, 2, n)
  Y = 2 - 3*X + epsilon
```

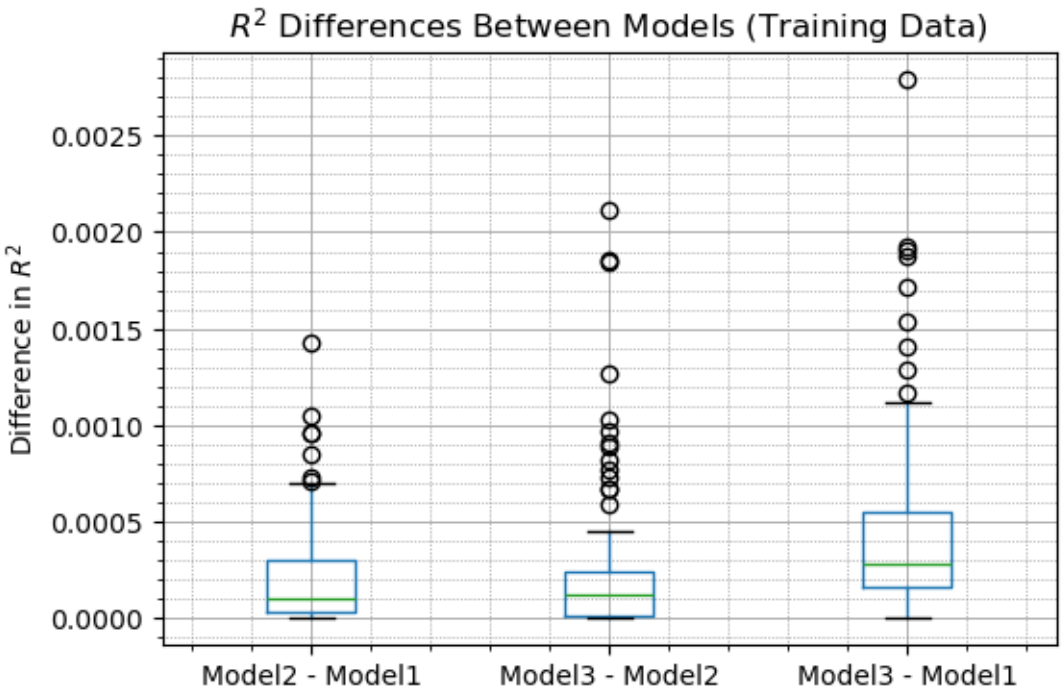
```
  X1 = X.reshape(-1, 1)
  X2 = np.column_stack((X, Z))
  X3 = np.column_stack((X, Z, np.sin(X)))

  model1 = LinearRegression().fit(X1, Y)
  model2 = LinearRegression().fit(X2, Y)
  model3 = LinearRegression().fit(X3, Y)

  r2_1 = model1.score(X1, Y)
  r2_2 = model2.score(X2, Y)
  r2_3 = model3.score(X3, Y)
  r2_1_test = model1.score(X1_test, Y_test)
  r2_2_test = model2.score(X2_test, Y_test)
  r2_3_test = model3.score(X3_test, Y_test)
  r2_values.append([r2_1, r2_2, r2_3])
  r2_values_test.append([r2_1_test, r2_2_test, r2_3_test])
r2_df = pd.DataFrame(r2_values, columns=['Model 1', 'Model 2', 'Model 3'])
r2_df_test = pd.DataFrame(r2_values_test, columns=['Model 1', 'Model 2', 'Model␣
 ↪3'])
```

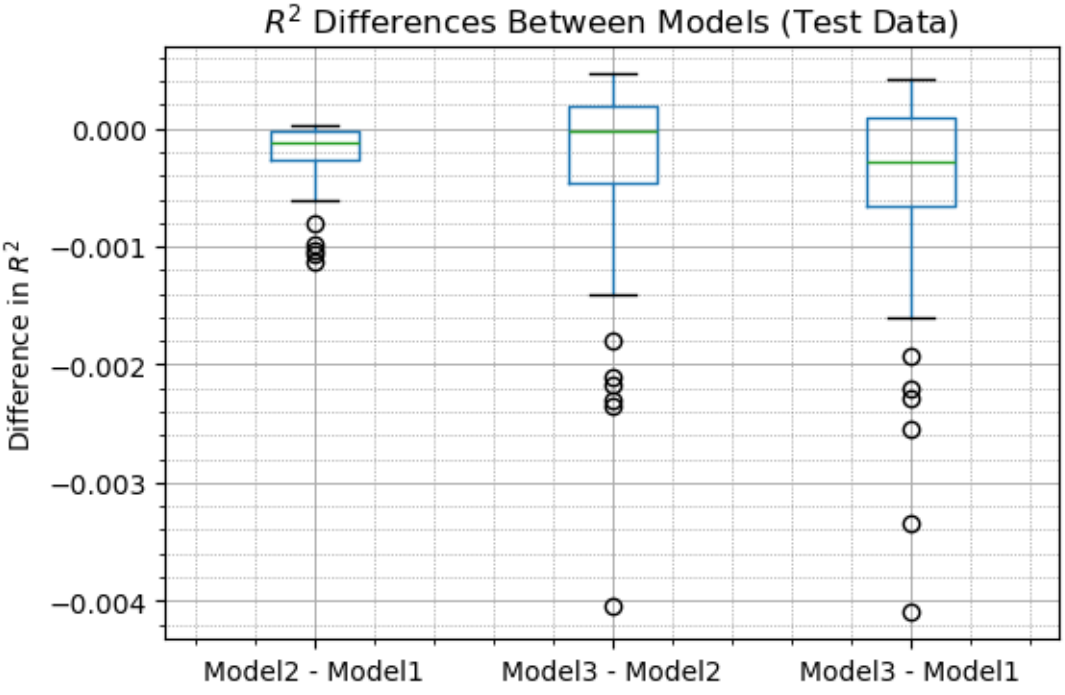
[5]:
```
# Plot for training data
fig, ax = plt.subplots(figsize=(6, 4))
r2_df.boxplot(ax=ax)
ax.set_ylabel('$R^2$')
ax.set_title('$R^2$ Values (Training Data)')
ax.grid(True, which='major', linestyle='-', linewidth='0.7')
ax.minorticks_on()
ax.grid(True, which='minor', linestyle=':', linewidth='0.5', color='gray')
plt.show()

# Plot for test data
fig, ax = plt.subplots(figsize=(6, 4))
r2_df_test.boxplot(ax=ax)
ax.set_ylabel('$R^2$')
ax.set_title('$R^2$ Values (Test Data)')
ax.grid(True, which='major', linestyle='-', linewidth='0.7')
ax.minorticks_on()
ax.grid(True, which='minor', linestyle=':', linewidth='0.5', color='gray')
plt.show()
```


R^2 Values (Training Data)
0.965
0.960
0.955
0.950
0.945
0.940
0.935
0.930
R^2
Model 1
Model 2
Model 3


R^2 Values (Test Data)
0.947
0.946
0.945
0.944
0.943
0.942
R^2
Model 1
Model 2
Model 3

[6]:
```
diff_21 = r2_df['Model 2'] - r2_df['Model 1']
diff_32 = r2_df['Model 3'] - r2_df['Model 2']
diff_31 = r2_df['Model 3'] - r2_df['Model 1']

diff_df = pd.DataFrame({
    'Model2 - Model1': diff_21,
    'Model3 - Model2': diff_32,
    'Model3 - Model1': diff_31
})

# Plot boxplots of the differences
fig, ax = plt.subplots(figsize=(6, 4))
diff_df.boxplot(ax=ax)
ax.set_ylabel('Difference in $R^2$')
ax.set_title('$R^2$ Differences Between Models (Training Data)')

# Add finer gridlines
ax.grid(True, which='major', linestyle='-', linewidth='0.7')
ax.minorticks_on()
ax.grid(True, which='minor', linestyle=':', linewidth='0.5', color='gray')

plt.show()

diff_21_test = r2_df_test['Model 2'] - r2_df_test['Model 1']
diff_32_test = r2_df_test['Model 3'] - r2_df_test['Model 2']
diff_31_test = r2_df_test['Model 3'] - r2_df_test['Model 1']

diff_df_test = pd.DataFrame({
    'Model2 - Model1': diff_21_test,
    'Model3 - Model2': diff_32_test,
    'Model3 - Model1': diff_31_test
})

# Plot boxplots of the differences
fig, ax = plt.subplots(figsize=(6, 4))
diff_df_test.boxplot(ax=ax)
ax.set_ylabel('Difference in $R^2$')
ax.set_title('$R^2$ Differences Between Models (Test Data)')

# Add finer gridlines
ax.grid(True, which='major', linestyle='-', linewidth='0.7')
ax.minorticks_on()
ax.grid(True, which='minor', linestyle=':', linewidth='0.5', color='gray')

plt.show()
```


R^2 Differences Between Models (Training Data)
Difference in R^2
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Model2 - Model1
Model3 - Model2
Model3 - Model1


R^2 Differences Between Models (Test Data)
Difference in R^2
0.000
−0.001
−0.002
−0.003
−0.004
Model2 - Model1
Model3 - Model2
Model3 - Model1

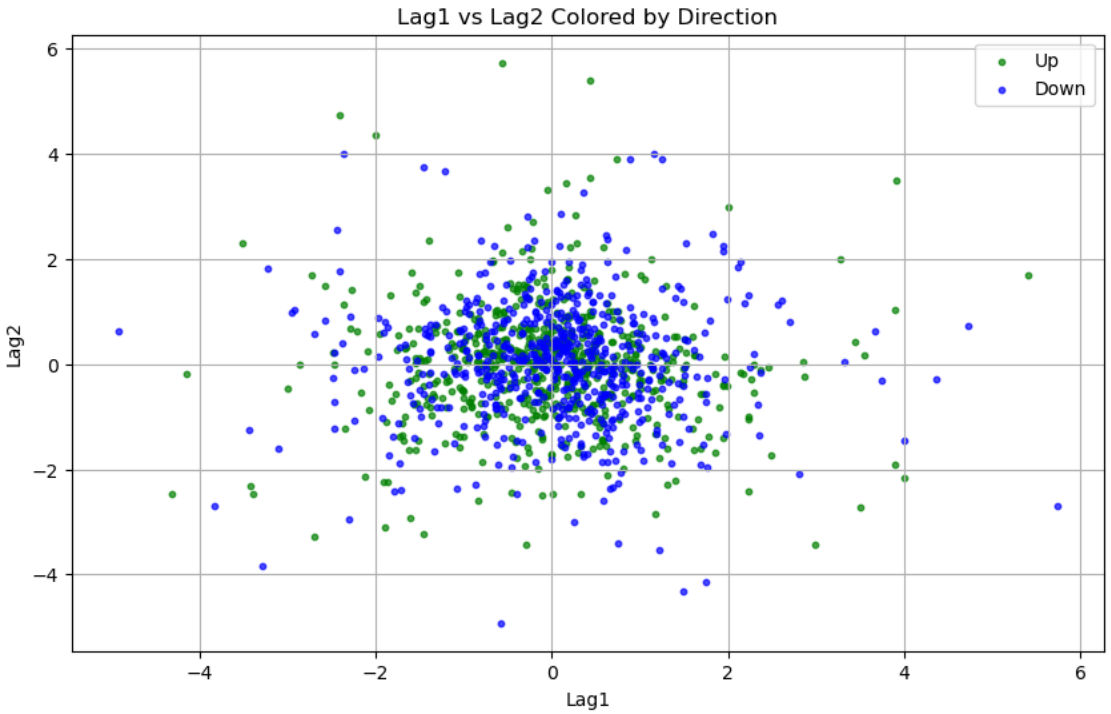
[7]:
```
##Question 2
```

[8]:
```
url = 'https://raw.githubusercontent.com/selva86/datasets/master/Smarket.csv'
smarket_data = pd.read_csv(url)
print(smarket_data.head())
```

```
   Year   Lag1   Lag2   Lag3   Lag4   Lag5  Volume  Today Direction
0  2001  0.381 -0.192 -2.624 -1.055  5.010  1.1913  0.959        Up
1  2001  0.959  0.381 -0.192 -2.624 -1.055  1.2965  1.032        Up
2  2001  1.032  0.959  0.381 -0.192 -2.624  1.4112 -0.623      Down
3  2001 -0.623  1.032  0.959  0.381 -0.192  1.2760  0.614        Up
4  2001  0.614 -0.623  1.032  0.959  0.381  1.2057  0.213        Up
```

[9]:
```
X = smarket_data[['Lag1', 'Lag2']]
Y = smarket_data['Direction']
```

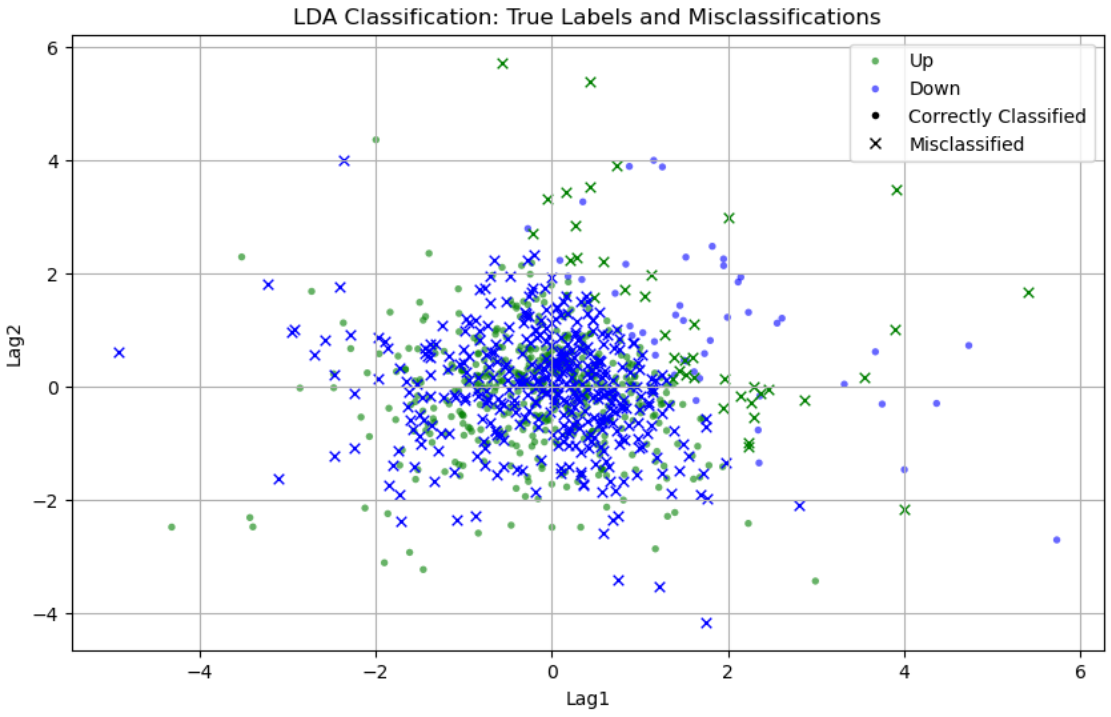
[10]:
```
color_map = {'Up': 'green', 'Down': 'blue'}
D = ['Up', 'Down']
plt.figure(figsize=(10, 6))
for direction in D:
    plt.scatter(X[Y == direction]['Lag1'],
                X[Y == direction]['Lag2'],
                label=direction,
                color=color_map[direction],
                alpha=0.7,
                s=10)
plt.xlabel('Lag1')
plt.ylabel('Lag2')
plt.title('Lag1 vs Lag2 Colored by Direction')
plt.legend()
plt.grid(True)
plt.show()
```



[11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.75,␣
↪random_state=42)
```

[12]:
```
# Question 2b, g
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
y_pred = lda.predict(X_train)
incorrect_pred = (y_pred != y_train.values)

plt.figure(figsize=(10, 6))
legends = []
for label in ['Up', 'Down']:
    idx = (y_train == label) & (~incorrect_pred)
    scatter = plt.scatter(X_train.loc[idx, 'Lag1'], X_train.loc[idx, 'Lag2'],
                c=color_map[label], label=f'{label}', alpha=0.6, s=15,␣
↪edgecolor='none')
    legends.append(scatter)

for label in ['Up', 'Down']:
    idx = (y_train == label) & (incorrect_pred)
    plt.scatter(X_train.loc[idx, 'Lag1'], X_train.loc[idx, 'Lag2'],
                c=color_map[label], marker='x', s=30, linewidths=1)

plt.xlabel('Lag1')
```

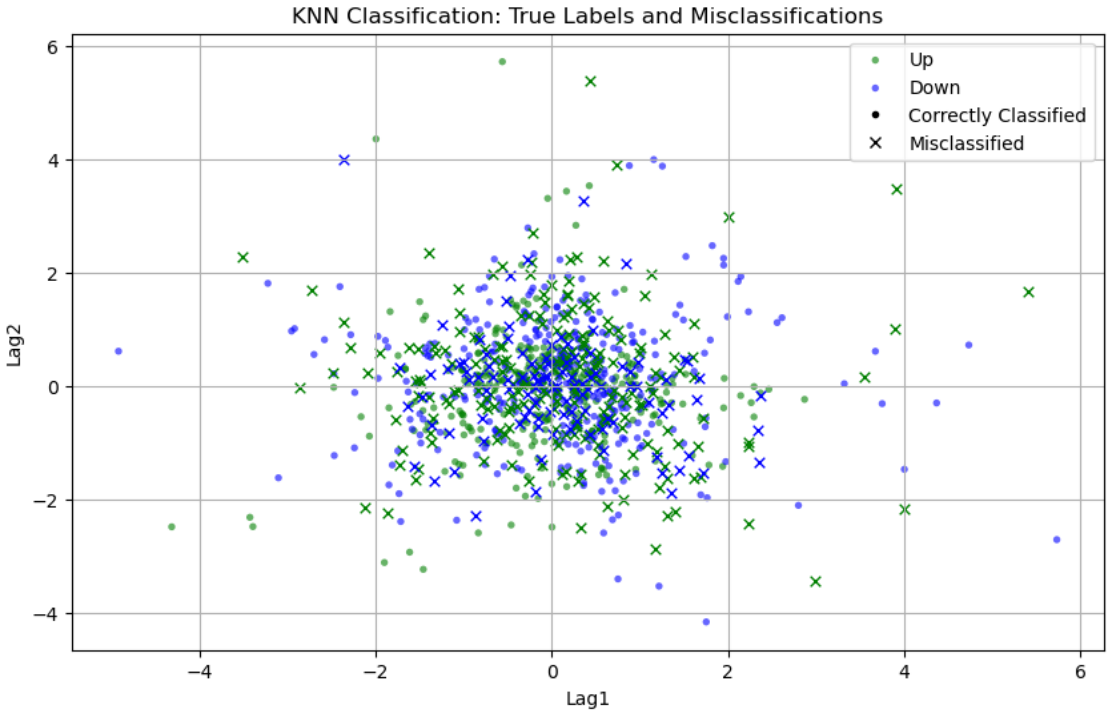
```
plt.ylabel('Lag2')
plt.title('LDA Classification: True Labels and Misclassifications')
correct_marker = mlines.Line2D([], [], color='black', marker='o',␣
 ↪linestyle='None', markersize=3, label='Correctly Classified')
misclassified_marker = mlines.Line2D([], [], color='black', marker='x',␣
 ↪linestyle='None', markersize=6, label='Misclassified')
plt.legend(handles=legends + [correct_marker, misclassified_marker], loc='best')

plt.grid(True)
plt.show()


accuracy = accuracy_score(y_train, y_pred)
print(f"Training Accuracy: {accuracy}")
print(f"Trainig Error: {1-accuracy}")

y_pred_test = lda.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test, labels=['Down', 'Up'])
TN, FP, FN, TP = cm.ravel()
accuracy = (TP + TN) / (TP + TN + FP + FN)
fnr = FN / (FN + TP)
fpr = FP / (FP + TN)

print(f"Test Accuracy: {accuracy:.4f}")
print(f"False Negative Rate (FNR): {fnr:.4f}")
print(f"False Positive Rate (FPR): {fpr:.4f}")
```



```
Training Accuracy: 0.5400213447171825
Trainig Error: 0.45997865528281745
Test Accuracy: 0.4760
False Negative Rate (FNR): 0.0952
False Positive Rate (FPR): 0.9036
```

[13]:
```python
# Question 2c, g
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_train)
incorrect_pred = (y_pred != y_train.values)

plt.figure(figsize=(10, 6))
legends = []
for label in ['Up', 'Down']:
    idx = (y_train == label) & (~incorrect_pred)
    scatter = plt.scatter(X_train.loc[idx, 'Lag1'], X_train.loc[idx, 'Lag2'],
                c=color_map[label], label=f'{label}', alpha=0.6, s=15,␣
 ↪edgecolor='none')
    legends.append(scatter)

for label in ['Up', 'Down']:
    idx = (y_train == label) & (incorrect_pred)
    plt.scatter(X_train.loc[idx, 'Lag1'], X_train.loc[idx, 'Lag2'],
                c=color_map[label], marker='x', s=30, linewidths=1)

plt.xlabel('Lag1')
plt.ylabel('Lag2')
plt.title('KNN Classification: True Labels and Misclassifications')
correct_marker = mlines.Line2D([], [], color='black', marker='o',␣
 ↪linestyle='None', markersize=3, label='Correctly Classified')
misclassified_marker = mlines.Line2D([], [], color='black', marker='x',␣
 ↪linestyle='None', markersize=6, label='Misclassified')
plt.legend(handles=legends + [correct_marker, misclassified_marker], loc='best')

plt.grid(True)
plt.show()

accuracy = accuracy_score(y_train, y_pred)
print(f"Training Accuracy: {accuracy}")
print(f"Trainig Error: {1-accuracy}")

y_pred_test = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test, labels=['Down', 'Up'])
```

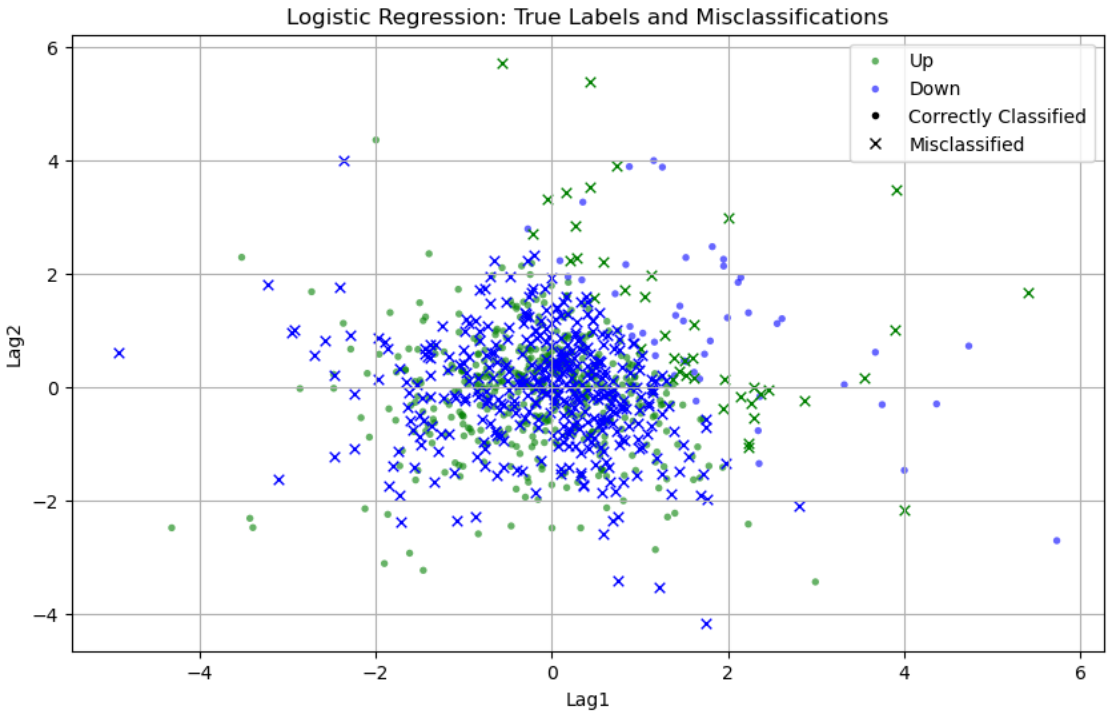
```
TN, FP, FN, TP = cm.ravel()
accuracy = (TP + TN) / (TP + TN + FP + FN)
fnr = FN / (FN + TP)
fpr = FP / (FP + TN)

print(f"Test Accuracy: {accuracy:.4f}")
print(f"False Negative Rate (FNR): {fnr:.4f}")
print(f"False Positive Rate (FPR): {fpr:.4f}")
```




```
Training Accuracy: 0.6456776947705443
Trainig Error: 0.35432230522945574
Test Accuracy: 0.4792
False Negative Rate (FNR): 0.5850
False Positive Rate (FPR): 0.4639
```

[14]:
```
# Question 2d, g
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_train)
incorrect_pred = (y_pred != y_train.values)

plt.figure(figsize=(10, 6))
legends = []
for label in ['Up', 'Down']:
```

```
    idx = (y_train == label) & (~incorrect_pred)
    scatter = plt.scatter(X_train.loc[idx, 'Lag1'], X_train.loc[idx, 'Lag2'],
                c=color_map[label], label=f'{label}', alpha=0.6, s=15,␣
 ↪edgecolor='none')
    legends.append(scatter)

for label in ['Up', 'Down']:
    idx = (y_train == label) & (incorrect_pred)
    plt.scatter(X_train.loc[idx, 'Lag1'], X_train.loc[idx, 'Lag2'],
                c=color_map[label], marker='x', s=30, linewidths=1)

plt.xlabel('Lag1')
plt.ylabel('Lag2')
plt.title('Logistic Regression: True Labels and Misclassifications')
correct_marker = mlines.Line2D([], [], color='black', marker='o',␣
 ↪linestyle='None', markersize=3, label='Correctly Classified')
misclassified_marker = mlines.Line2D([], [], color='black', marker='x',␣
 ↪linestyle='None', markersize=6, label='Misclassified')
plt.legend(handles=legends + [correct_marker, misclassified_marker], loc='best')

plt.grid(True)
plt.show()


accuracy = accuracy_score(y_train, y_pred)
print(f"Training Accuracy: {accuracy}")
print(f"Trainig Error: {1-accuracy}")

y_pred_test = log_reg.predict(X_test)
cm = confusion_matrix(y_test, y_pred_test, labels=['Down', 'Up'])

TN, FP, FN, TP = cm.ravel()
accuracy = (TP + TN) / (TP + TN + FP + FN)
fnr = FN / (FN + TP)
fpr = FP / (FP + TN)

print(f"Test Accuracy: {accuracy:.4f}")
print(f"False Negative Rate (FNR): {fnr:.4f}")
print(f"False Positive Rate (FPR): {fpr:.4f}")
```



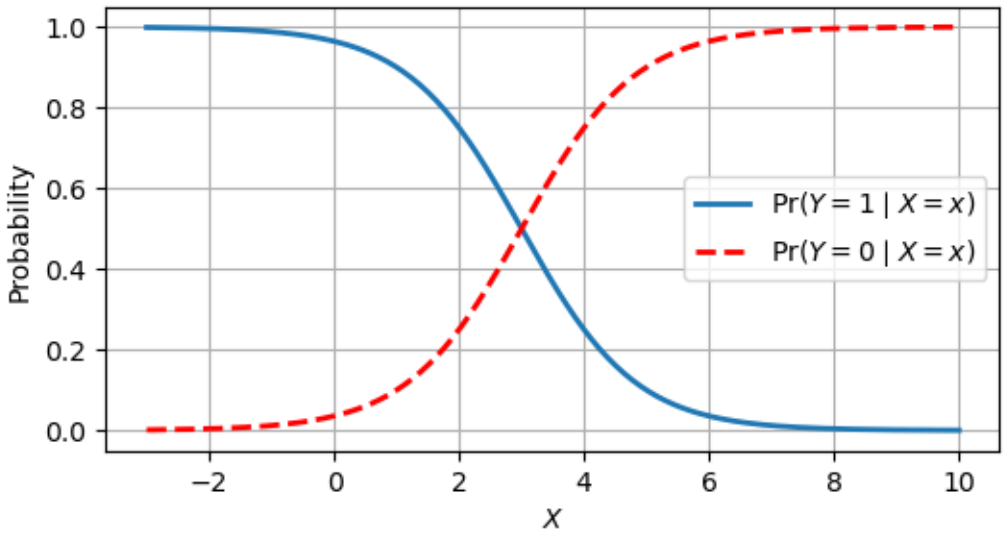

```
Training Accuracy: 0.5389541088580576
Trainig Error: 0.4610458911419424
Test Accuracy: 0.4792
False Negative Rate (FNR): 0.0952
False Positive Rate (FPR): 0.8976
```

[15]:
```
##Question 4
```

[16]:
```
#Question 4b
beta_0 = 3*np.log(9)/2
beta_1 = -np.log(9)/2

def p_y1(x):
    return np.exp(beta_0 + beta_1 * x) / (1 + np.exp(beta_0 + beta_1 * x))

def p_y0(x):
    return 1 - p_y1(x)

np.random.seed(50)
x_values = np.linspace(-3, 10, 200)

p1_values = p_y1(x_values)
p0_values = p_y0(x_values)
```

```
plt.figure(figsize=(6, 3))
plt.plot(x_values, p1_values, label='$\Pr(Y=1 \mid X=x)$', linewidth=2)
plt.plot(x_values, p0_values, label='$\Pr(Y=0 \mid X=x)$', color='red',␣
 ↪linestyle='--', linewidth=2)

plt.xlabel('$X$')
plt.ylabel('Probability')
plt.legend()
plt.grid(True)
plt.ylim(-0.05, 1.05)
plt.show()
```

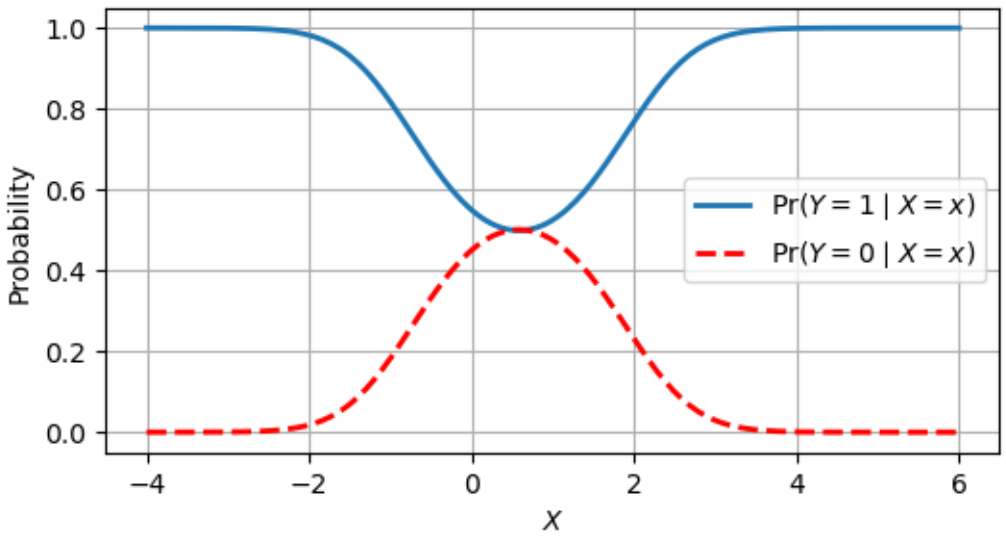
```
<>:18: SyntaxWarning: invalid escape sequence '\P'
<>:19: SyntaxWarning: invalid escape sequence '\P'
<>:18: SyntaxWarning: invalid escape sequence '\P'
<>:19: SyntaxWarning: invalid escape sequence '\P'
/var/folders/pk/90dn4p5556l45snxl195_hlm0000gn/T/ipykernel_78619/1917992748.py:1
8: SyntaxWarning: invalid escape sequence '\P'
  plt.plot(x_values, p1_values, label='$\Pr(Y=1 \mid X=x)$', linewidth=2)
/var/folders/pk/90dn4p5556l45snxl195_hlm0000gn/T/ipykernel_78619/1917992748.py:1
9: SyntaxWarning: invalid escape sequence '\P'
  plt.plot(x_values, p0_values, label='$\Pr(Y=0 \mid X=x)$', color='red',
linestyle='--', linewidth=2)
```



[17]:
```
#Question 5

beta_0 = 0.2
beta_1 = -0.7
```

```
beta_2 = 0.6

def p_y1(x):
    return np.exp(beta_0 + beta_1 * x + beta_2 * x**2) / (1 + np.exp(beta_0 +␣
 ↪beta_1 * x + beta_2 * x**2))

def p_y0(x):
    return 1 - p_y1(x)

np.random.seed(51)
x_values = np.linspace(-4, 6, 200)

p1_values = p_y1(x_values)
p0_values = p_y0(x_values)

plt.figure(figsize=(6, 3))
plt.plot(x_values, p1_values, label='$\Pr(Y=1 \mid X=x)$', linewidth=2)
plt.plot(x_values, p0_values, label='$\Pr(Y=0 \mid X=x)$', color='red',␣
 ↪linestyle='--', linewidth=2)

plt.xlabel('$X$')
plt.ylabel('Probability')
plt.legend()
plt.grid(True)
plt.ylim(-0.05, 1.05)
plt.show()
```

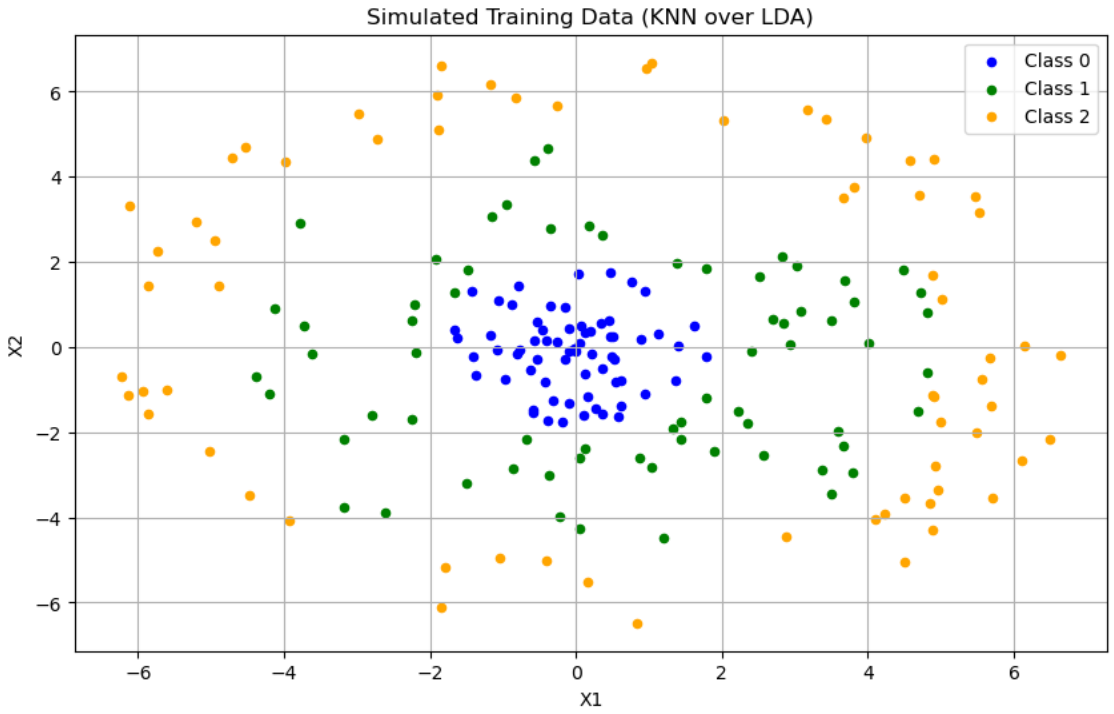
```
<>:20: SyntaxWarning: invalid escape sequence '\P'
<>:21: SyntaxWarning: invalid escape sequence '\P'
<>:20: SyntaxWarning: invalid escape sequence '\P'
<>:21: SyntaxWarning: invalid escape sequence '\P'
/var/folders/pk/90dn4p5556l45snxl195_hlm0000gn/T/ipykernel_78619/1801566761.py:2
0: SyntaxWarning: invalid escape sequence '\P'
  plt.plot(x_values, p1_values, label='$\Pr(Y=1 \mid X=x)$', linewidth=2)
/var/folders/pk/90dn4p5556l45snxl195_hlm0000gn/T/ipykernel_78619/1801566761.py:2
1: SyntaxWarning: invalid escape sequence '\P'
  plt.plot(x_values, p0_values, label='$\Pr(Y=0 \mid X=x)$', color='red',
linestyle='--', linewidth=2)
```



[18]:
```
##Question 6
```

[19]:
```
# Question 6a
np.random.seed(21)
n_train = 200
n_test = 2000

r_0_train = np.random.uniform(0, 2, size=66)
theta_0_train = np.random.uniform(0, 2 * np.pi, size=66)
X0_train = np.column_stack([r_0_train * np.cos(theta_0_train), r_0_train * np.
 ↪sin(theta_0_train)])

r_1_train = np.random.uniform(2, 5, size=67)
theta_1_train = np.random.uniform(0, 2 * np.pi, size=67)
X1_train = np.column_stack([r_1_train * np.cos(theta_1_train), r_1_train * np.
 ↪sin(theta_1_train)])

r_2_train = np.random.uniform(5, 7, size=67)
theta_2_train = np.random.uniform(0, 2 * np.pi, size=67)
X2_train = np.column_stack([r_2_train * np.cos(theta_2_train), r_2_train * np.
 ↪sin(theta_2_train)])

X_train = np.vstack([X0_train, X1_train, X2_train])
y_train = np.array([0]*(66) + [1]*(67) + [2]*(67))

r_0_test = np.random.uniform(0, 2, size=666)
theta_0_test = np.random.uniform(0, 2 * np.pi, size=666)
```

```
X0_test = np.column_stack([r_0_test * np.cos(theta_0_test), r_0_test * np.
 ↪sin(theta_0_test)])

r_1_test = np.random.uniform(2, 5, size=667)
theta_1_test = np.random.uniform(0, 2 * np.pi, size=667)
X1_test = np.column_stack([r_1_test * np.cos(theta_1_test), r_1_test * np.
 ↪sin(theta_1_test)])

r_2_test = np.random.uniform(5, 7, size=667)
theta_2_test = np.random.uniform(0, 2 * np.pi, size=667)
X2_test = np.column_stack([r_2_test * np.cos(theta_2_test), r_2_test * np.
 ↪sin(theta_2_test)])

X_test = np.vstack([X0_test, X1_test, X2_test])
y_test = np.array([0]*(666) + [1]*(667) + [2]*(667))

color_map = {1: 'green', 0: 'blue', 2: 'orange'}
plt.figure(figsize=(10, 6))
for cls in [0, 1, 2]:
    plt.scatter(X_train[y_train == cls][:,0],
                X_train[y_train == cls][:,1],
                label=f'Class {cls}',
                color=color_map[cls],
                s=20)
plt.title('Simulated Training Data (KNN over LDA)')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.grid(True)
plt.show()
```



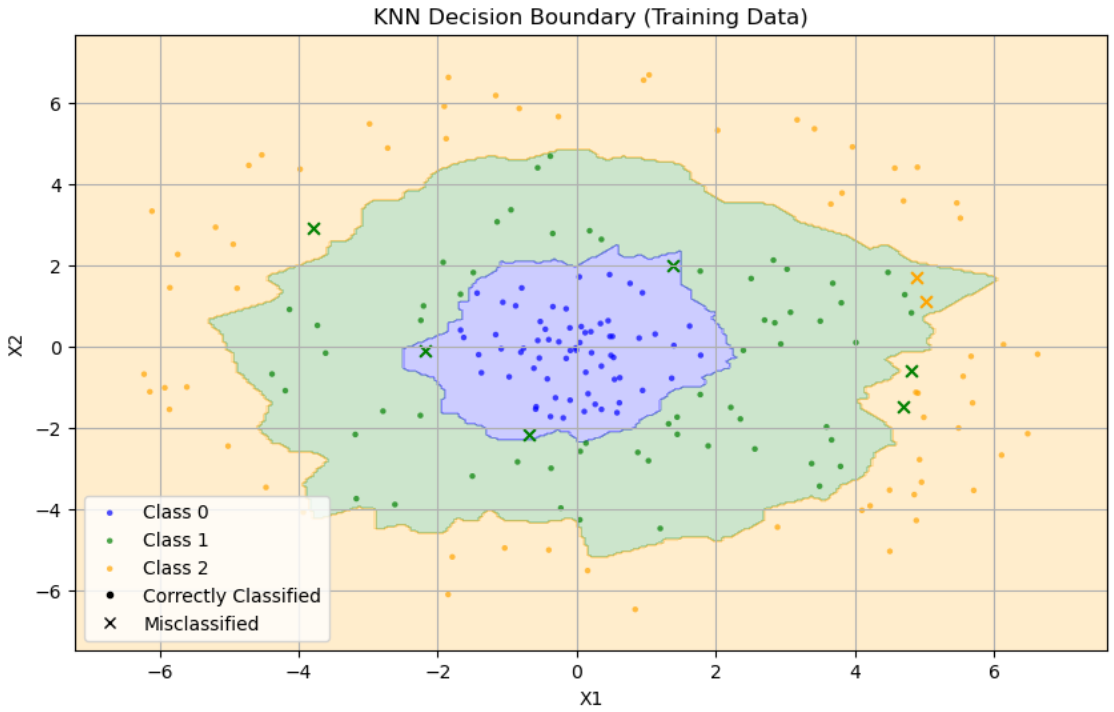
[20]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"KNN Accuracy: {accuracy}")
print(f"KNN Error: {1-accuracy}")

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
y_pred = lda.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"LDA Accuracy: {accuracy}")
print(f"LDA Error: {1-accuracy}")
```

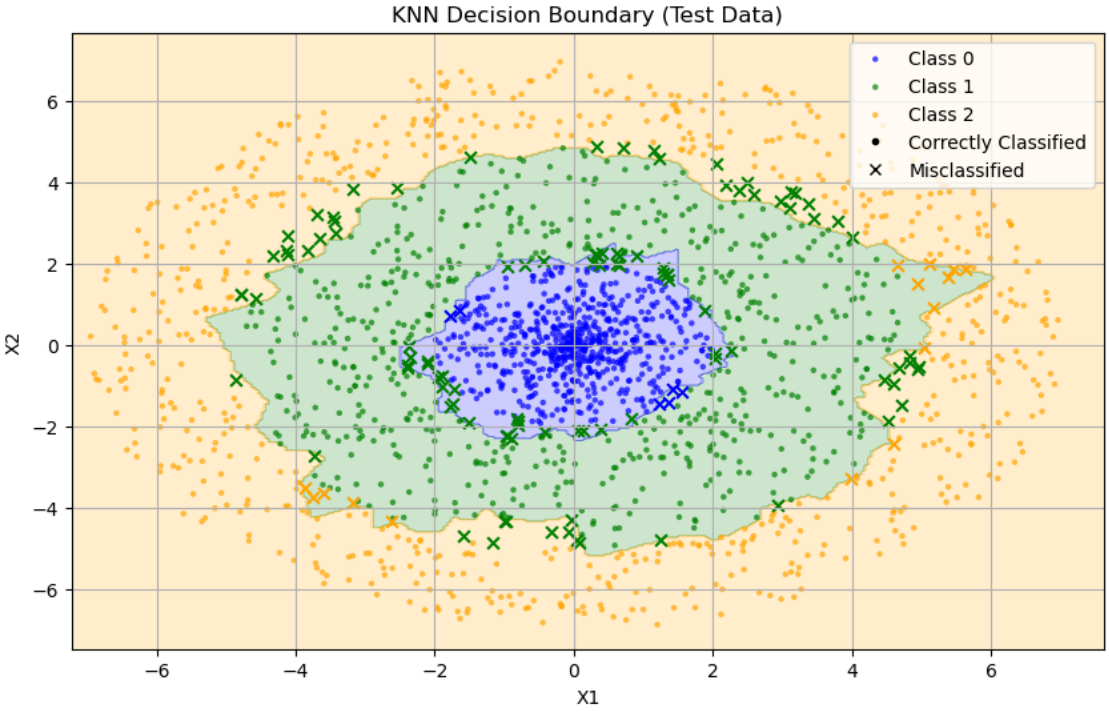
```
KNN Accuracy: 0.939
KNN Error: 0.061000000000000054
LDA Accuracy: 0.3895
LDA Error: 0.6105
```

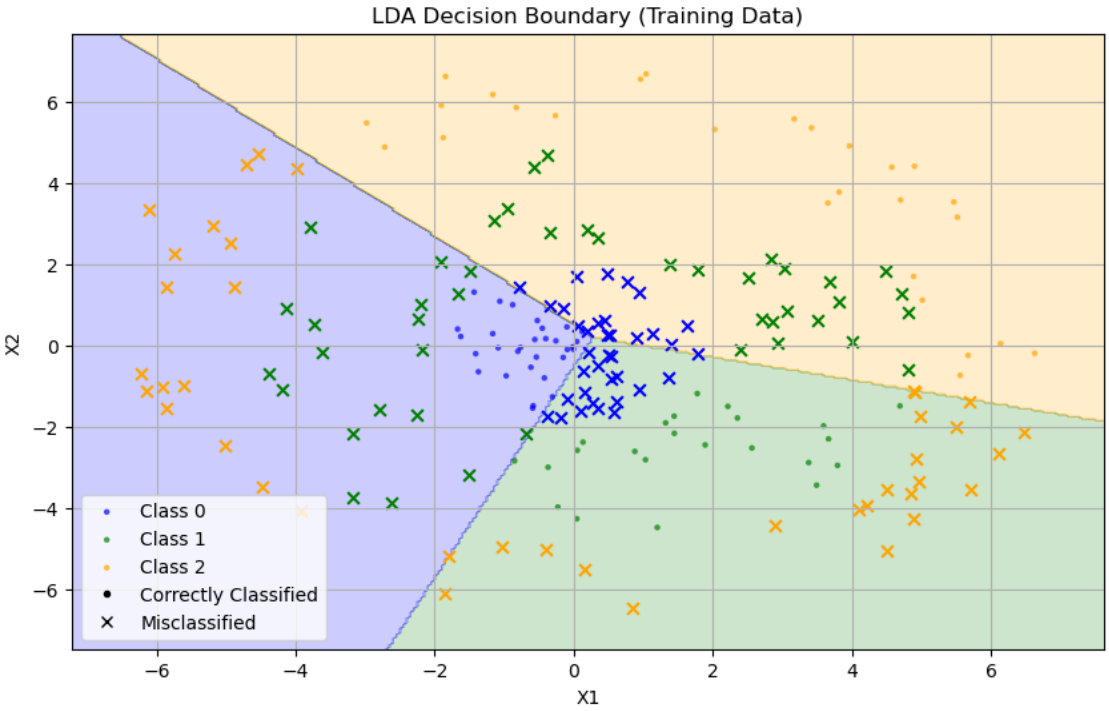
[21]:
```
# Create meshgrid
h = 0.05
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
```

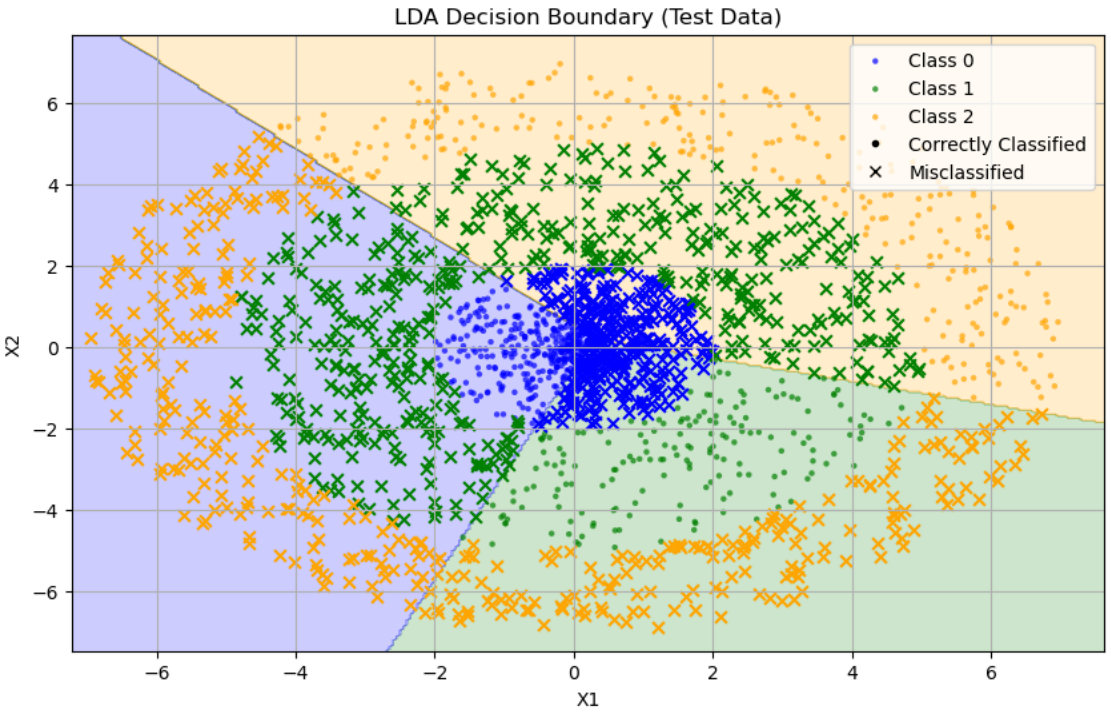
```
                     np.arange(y_min, y_max, h))

Z_knn = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z_knn = Z_knn.reshape(xx.shape)

Z_lda = lda.predict(np.c_[xx.ravel(), yy.ravel()])
Z_lda = Z_lda.reshape(xx.shape)

from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['blue', 'green', 'orange'])
color_map = {1: 'green', 0: 'blue', 2: 'orange'}
def plot_with_boundary(X_data, y_data, model, boundary, title):
    legends = []
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, boundary, alpha=0.2, cmap=custom_cmap)
    y_pred = model.predict(X_data)

    correct = (y_data == y_pred)
    incorrect = (y_data != y_pred)
    for label in [0, 1, 2]:
      idx = (y_data == label) & (~incorrect)
      scatter = plt.scatter(X_data[idx, 0], X_data[idx, 1],
                  c=color_map[label], label=f'Class {label}', alpha=0.7, s=10,␣
 ↪edgecolor='none')
      legends.append(scatter)

    for label in [0, 1, 2]:
        idx = (y_data == label) & (incorrect)
        plt.scatter(X_data[idx, 0], X_data[idx, 1],
                    c=color_map[label], marker='x', s=40, linewidths=1.5)

    plt.title(title)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.grid(True)
    # plt.legend()
    correct_marker = mlines.Line2D([], [], color='black', marker='o',␣
 ↪linestyle='None', markersize=3, label='Correctly Classified')
    misclassified_marker = mlines.Line2D([], [], color='black', marker='x',␣
 ↪linestyle='None', markersize=6, label='Misclassified')
    plt.legend(handles=legends + [correct_marker, misclassified_marker],␣
 ↪loc='best')
    plt.show()

plot_with_boundary(X_train, y_train, knn, Z_knn, 'KNN Decision Boundary␣
 ↪(Training Data)')
```

```
plot_with_boundary(X_test, y_test, knn, Z_knn, 'KNN Decision Boundary (Test␣
 ↪Data)')
plot_with_boundary(X_train, y_train, lda, Z_lda, 'LDA Decision Boundary␣
 ↪(Training Data)')
plot_with_boundary(X_test, y_test, lda, Z_lda, 'LDA Decision Boundary (Test␣
 ↪Data)')
```



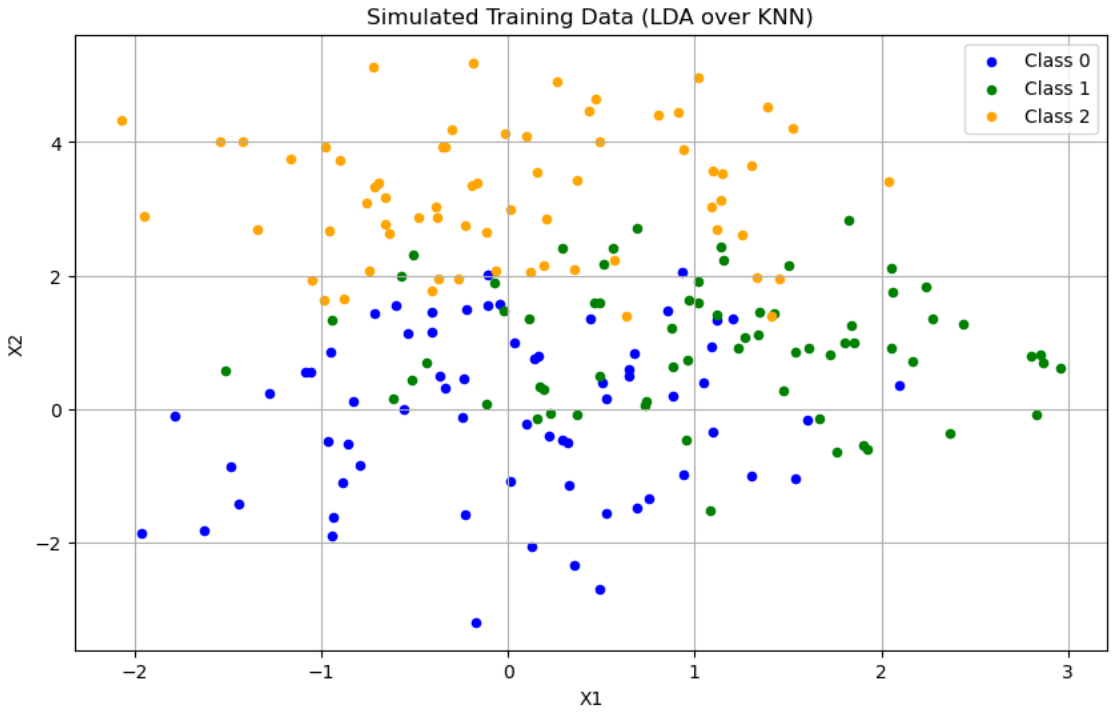

KNN Decision Boundary (Test Data)
Class 0
Class 1
Class 2
Correctly Classified
Misclassified
X1
X2


LDA Decision Boundary (Training Data)
Class 0
Class 1
Class 2
Correctly Classified
Misclassified
X1
X2



[22]:
```
# Question 6b
np.random.seed(20)
n_train = 200
n_test = 2000

X0_train = np.random.normal(loc=[0, 0], scale=1, size=(66, 2))
X1_train = np.random.normal(loc=[1, 1], scale=1, size=(67, 2))
X2_train = np.random.normal(loc=[0, 3], scale=1, size=(67, 2))

X_train = np.vstack([X0_train, X1_train, X2_train])
y_train = np.array([0]*(66) + [1]*(67) + [2]*(67))

X0_test = np.random.normal(loc=[0, 0], scale=1, size=(666, 2))
X1_test = np.random.normal(loc=[1, 1], scale=1, size=(667, 2))
X2_test = np.random.normal(loc=[0, 3], scale=1, size=(667, 2))

X_test = np.vstack([X0_test, X1_test, X2_test])
y_test = np.array([0]*(666) + [1]*(667) + [2]*(667))
color_map = {1: 'green', 0: 'blue', 2: 'orange'}
plt.figure(figsize=(10, 6))
for cls in [0, 1, 2]:
    plt.scatter(X_train[y_train == cls][:,0],
                X_train[y_train == cls][:,1],
                label=f'Class {cls}',
```

```
                color=color_map[cls],
                s=20)

plt.title('Simulated Training Data (LDA over KNN)')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.grid(True)
plt.show()
```

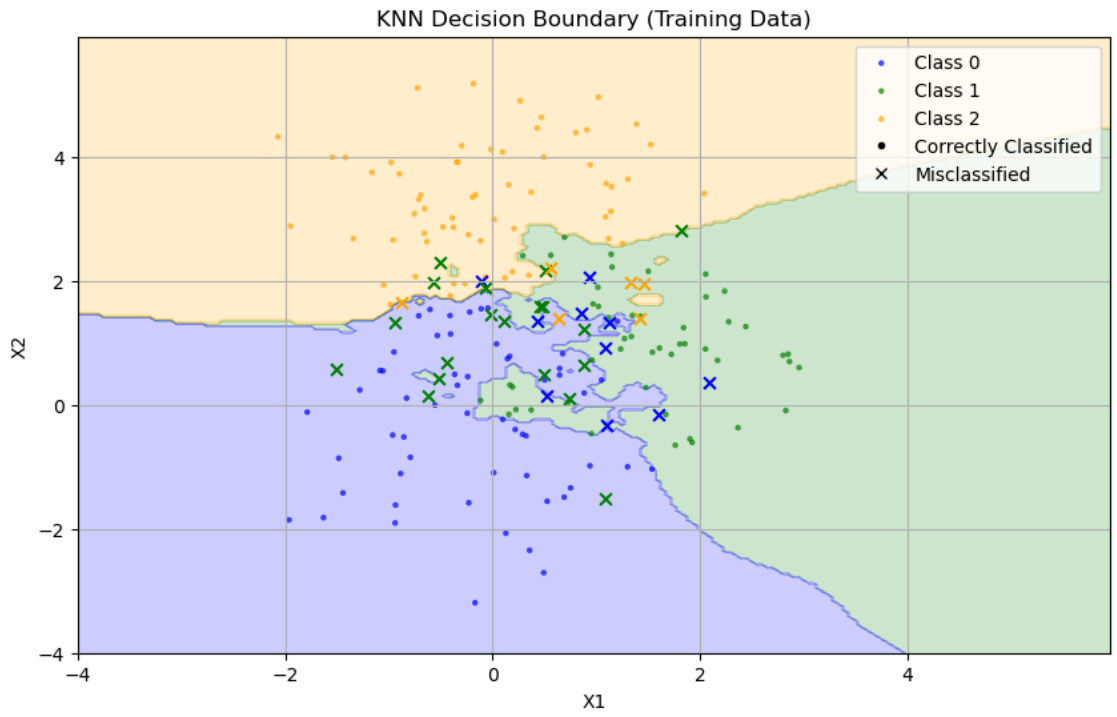


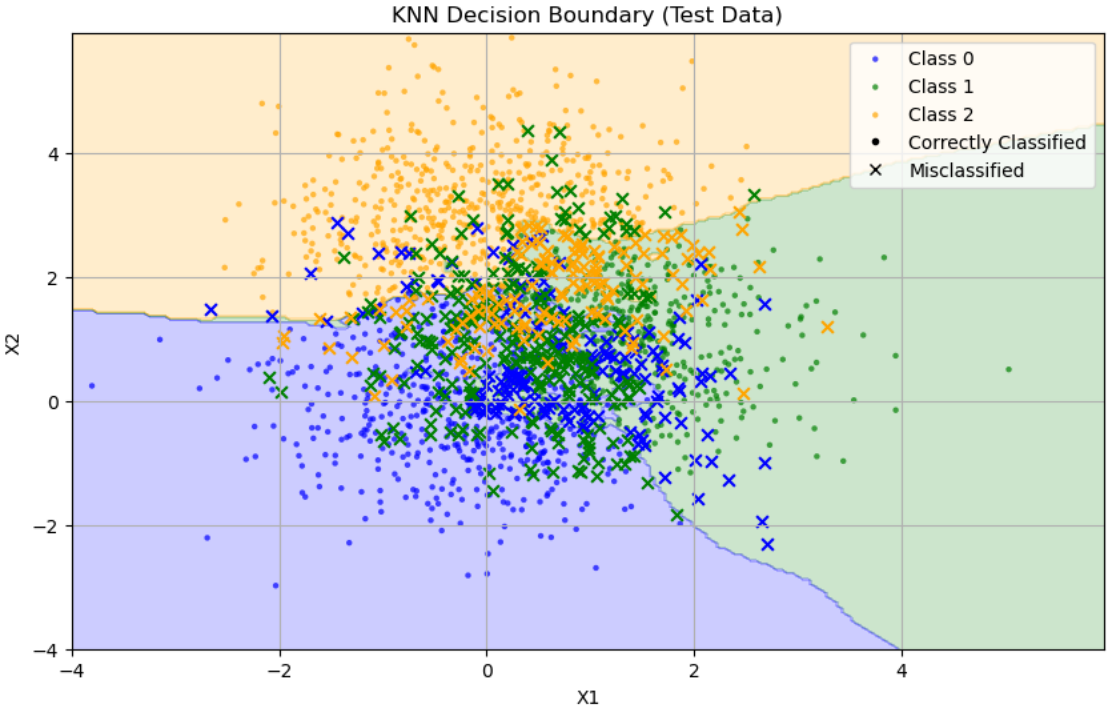
[23]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"KNN Accuracy: {accuracy}")
print(f"KNN Error: {1-accuracy}")

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
y_pred = lda.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"LDA Accuracy: {accuracy}")
print(f"LDA Error: {1-accuracy}")
```

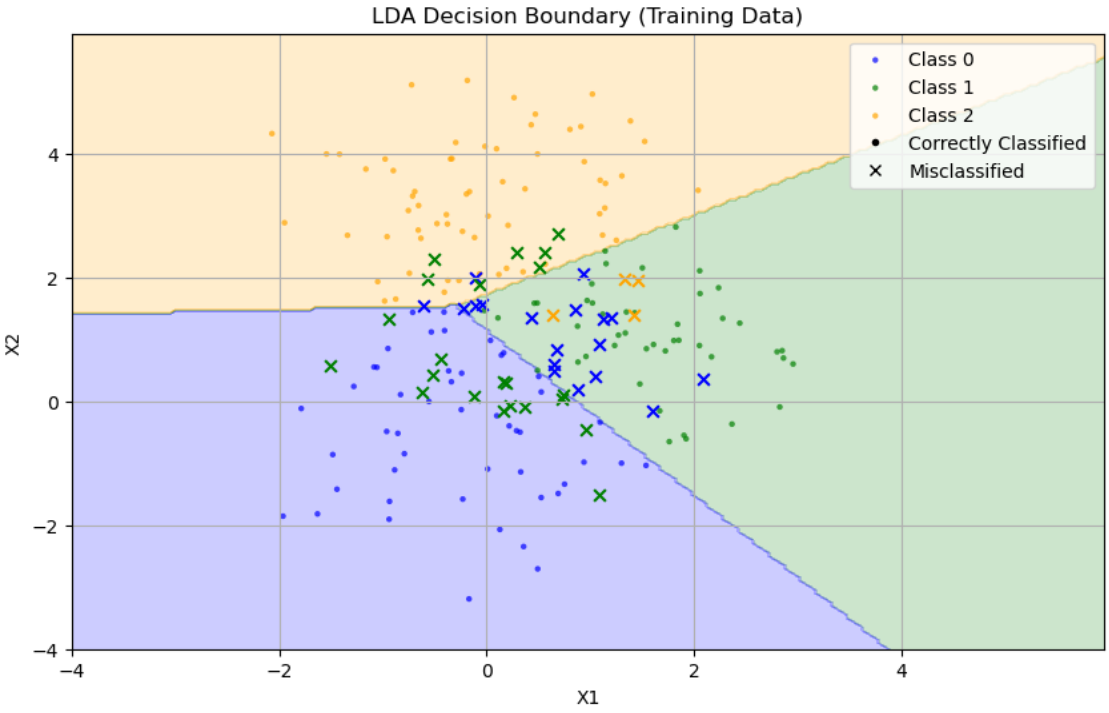
```
KNN Accuracy: 0.714
KNN Error: 0.28600000000000003
LDA Accuracy: 0.7555
LDA Error: 0.24450000000000005
```

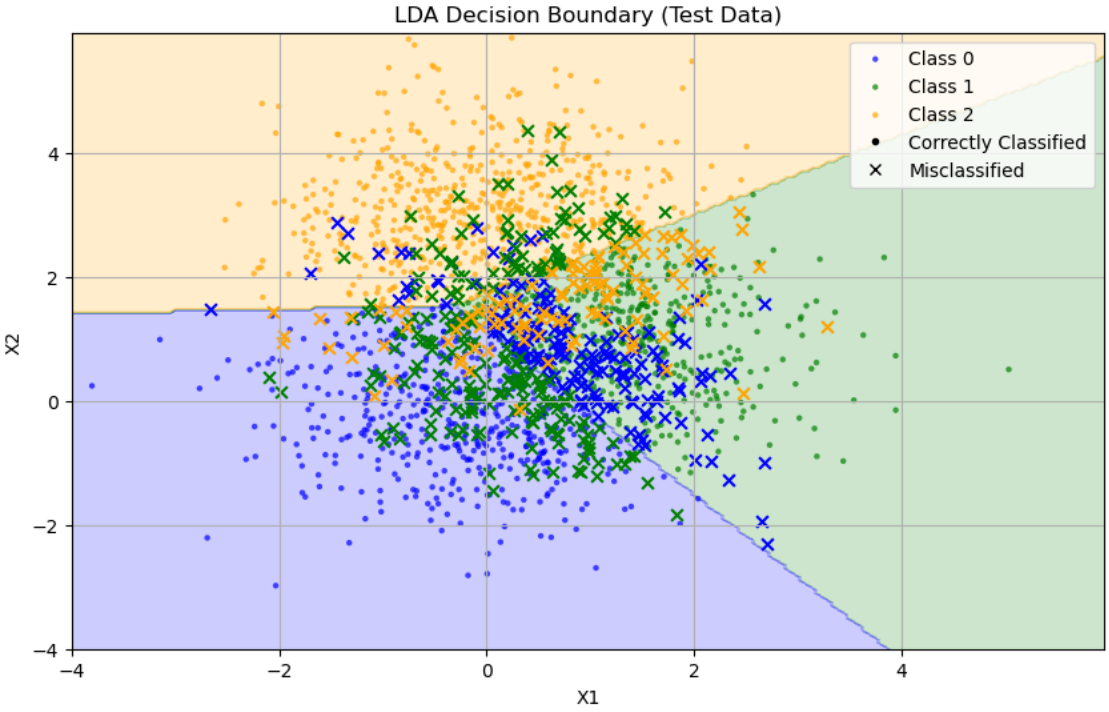
[24]:
```
# Create meshgrid
h = 0.05
x_min, x_max = -4,6
y_min, y_max = -4,6
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

Z_knn = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z_knn = Z_knn.reshape(xx.shape)

Z_lda = lda.predict(np.c_[xx.ravel(), yy.ravel()])
Z_lda = Z_lda.reshape(xx.shape)

custom_cmap = ListedColormap(['blue', 'green', 'orange'])
color_map = {1: 'green', 0: 'blue', 2: 'orange'}
def plot_with_boundary(X_data, y_data, model, boundary, title):
    legends = []
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, boundary, alpha=0.2, cmap=custom_cmap)
    y_pred = model.predict(X_data)

    correct = (y_data == y_pred)
    incorrect = (y_data != y_pred)

    for label in [0, 1, 2]:
      idx = (y_data == label) & (~incorrect)
      scatter = plt.scatter(X_data[idx, 0], X_data[idx, 1],
                  c=color_map[label], label=f'Class {label}', alpha=0.7, s=10,␣
 ↪edgecolor='none')
      legends.append(scatter)

    for label in [0, 1, 2]:
        idx = (y_data == label) & (incorrect)
        plt.scatter(X_data[idx, 0], X_data[idx, 1],
                    c=color_map[label], marker='x', s=40, linewidths=1.5)

    plt.title(title)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.grid(True)
    # plt.legend()
```

```
    correct_marker = mlines.Line2D([], [], color='black', marker='o',␣
 ↪linestyle='None', markersize=3, label='Correctly Classified')
    misclassified_marker = mlines.Line2D([], [], color='black', marker='x',␣
 ↪linestyle='None', markersize=6, label='Misclassified')
    plt.legend(handles=legends + [correct_marker, misclassified_marker],␣
 ↪loc='best')
    plt.show()

plot_with_boundary(X_train, y_train, knn, Z_knn, 'KNN Decision Boundary␣
 ↪(Training Data)')
plot_with_boundary(X_test, y_test, knn, Z_knn, 'KNN Decision Boundary (Test␣
 ↪Data)')
plot_with_boundary(X_train, y_train, lda, Z_lda, 'LDA Decision Boundary␣
 ↪(Training Data)')
plot_with_boundary(X_test, y_test, lda, Z_lda, 'LDA Decision Boundary (Test␣
 ↪Data)')
```




KNN Decision Boundary (Test Data)
Class 0
Class 1
Class 2
Correctly Classified
Misclassified
X1
X2
−4
−2
0
2
4


LDA Decision Boundary (Training Data)
Class 0
Class 1
Class 2
Correctly Classified
Misclassified
X1
X2
−4
−2
0
2
4


LDA Decision Boundary (Test Data)
Class 0
Class 1
Class 2
Correctly Classified
Misclassified
X1
X2
−4
−2
0
2
4
−4
−2
0
2
4

[ ]: