**27. Python Linear Discriminant Analysis (LDA) for Predicting Shipping Outcomes in a Warehouse**

After discussing logistic regression, I now move on to using classifiers from scikit-learn. Today, I will focus on fitting a Linear Discriminant Analysis (LDA) model. In scikit-learn, the workflow for prediction problems involves creating an object for the classifier—in this case, LDA is a shorthand for LinearDiscriminantAnalysis. At first, the classifier object is initialized without any data, only specifying certain arguments. For example, I might choose to store the covariance matrix in LDA in case I want to inspect it later. After initializing the estimator, I fit it to a matrix of features (X) and a response variable (y). This fitting procedure allows me to use the model to predict new data or the original training data. This approach is quite standard across many scikit-learn estimators.

As usual, I begin by splitting the dataset into training and test sets. I use the same design matrices as before for training and testing, but I drop the intercept column this time. LDA does not require the intercept column, and including it would cause the covariance matrix to be non-invertible, leading to an error. After fitting the LDA model to the training data, I can examine several key parameters, such as the common covariance, within-class means, and the class labels. The scaling matrix, in particular, is critical as it represents the discriminant function used to predict whether a shipment will be "On Time" or "Late."

To evaluate the model's performance, I use the predict method on the test data. This method is standard for most scikit-learn classifiers. After making predictions, I construct a confusion matrix using the predicted and actual test labels. This matrix helps visualize the accuracy of the classifier by showing the distribution of true positives, false positives, true negatives, and false negatives.

While the typical threshold for assigning classes is 50%, I can easily adjust this threshold to change the balance of predicted outcomes. The LDA model provides estimated class probabilities for the test data, with the first column representing the probability of a shipment being "On Time" and the second column representing "Late." By increasing the threshold, I classify fewer observations as "On Time" and more as "Late," adjusting the off-diagonal errors in the confusion matrix. Modifying this threshold can help improve accuracy depending on the proportion of "On Time" and "Late" shipments and the within-class accuracy.

Moving on to the next classifier, **Quadratic Discriminant Analysis (QDA)**, the main difference from LDA is that QDA uses a class-specific covariance matrix. As promised, the code for fitting QDA is nearly identical to that of LDA. I use the same training features and labels, and I call the fit method on an instance of the QuadraticDiscriminantAnalysis classifier. Like with LDA, I can examine the covariance for specific classes. For example, to inspect the covariance for the "Late" class, I look at the first entry of the covariance matrix returned.

Since QDA involves a separate covariance matrix for each class, the classification function becomes quadratic rather than linear. The discriminant function for QDA is not described solely by the scaling matrix; it also depends on the covariance and the difference in means. In LDA, the scaling matrix effectively represents the difference in means, multiplied by the inverse of the common covariance matrix.

To evaluate QDA's performance, I again use the predict method on the test features and generate a confusion matrix. The QDA model achieves about 60% accuracy, which is notably better than the 55% accuracy I obtained with logistic regression for these features. This result is promising, particularly for complex shipping data.

Having observed this pattern with two classifiers, I can see how scikit-learn provides a consistent and straightforward approach to fitting various classifiers. I then proceed to another classifier, **Naive Bayes**. From the lectures or textbook, I recall that for continuous features, Naive Bayes is closely related to QDA but imposes a simpler structure. It assumes that the covariance matrices are diagonal for each class, reducing the number of parameters to estimate. This simplification makes Naive Bayes a more constrained version of QDA.

Fitting Naive Bayes follows the same procedure as LDA and QDA. I call the fit method on the training features and labels and evaluate the model similarly. Naive Bayes assumes diagonal covariance matrices, which means I only need to store the variance of each feature within each class rather than the full covariance matrix. For example, the variance\_ attribute stores the variances of the relevant shipping features (Feature1 and Feature2) for both classes, "On Time" and "Late."

To check the performance of Naive Bayes, I construct a confusion matrix similarly to LDA and QDA. The Naive Bayes model achieves an accuracy of around 59%, close to LDA's performance. All these discriminant analysis methods provide reasonably good results for predicting shipping outcomes, demonstrating their effectiveness in handling complex data.

The next step will be to explore another classifier, **k-Nearest Neighbors (k-NN)**, to further compare their performance on predicting shipping outcomes.