Learning Linear Discriminant Analysis (LDA) with Python: A Step-by-Step Guide

Authored by

Mohammed looti

November 6, 2025

Linear Discriminant Analysis (LDA) is a venerable and powerful technique fundamental to statistical modeling and modern machine learning. Its core objective is to determine a linear combination of features that optimally separates two or more predefined classes of observations. Unlike complex non-linear classifiers, LDA provides an interpretable mechanism for both dimensionality reduction and high-efficiency classification.

This methodology proves invaluable when dealing with a dataset containing a collection of **predictor variables** and a categorical response variable that needs assignment into distinct groups. Crucially, LDA operates by maximizing the distance between class means while simultaneously minimizing the variance or spread of data points within each specific class, resulting in robust classification boundaries.

This comprehensive tutorial offers a rigorous, step-by-step guide detailing the implementation and robust evaluation of **Linear Discriminant Analysis** using the powerful Python ecosystem, with a specific focus on leveraging the functionality provided by the widely adopted **scikit-learn** library.

Step 1: Establishing the Python Environment and Required Libraries

The foundation of any successful data science endeavor lies in properly configuring the required environment and importing essential tools. For this particular analysis, we rely on several foundational Python libraries. These include **scikit-learn**, which provides the core LDA model and necessary cross-validation utilities; Pandas, used for efficient data manipulation and structuring; and **NumPy**, utilized for high-performance numerical operations crucial to vector processing.

The following code snippet loads all required modules and functions into our workspace. This step ensures we have immediate access to classes needed for model selection, performance evaluation, and the central `LinearDiscriminantAnalysis` estimator itself, enabling a seamless transition into data processing and modeling:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Step 2: Data Acquisition and Preparation using the Iris Dataset

To effectively demonstrate the capabilities of LDA, we will utilize the historically significant **Iris dataset**. This dataset is a standard resource for classification and clustering examples and is conveniently bundled within the **scikit-learn** library, eliminating the need for external downloads and ensuring immediate access to clean data.

After loading the raw dataset, the immediate next step involves converting the structured data into a highly readable and manipulable Pandas DataFrame. This crucial conversion step allows for clear definition and labeling of feature names and the target variable, making subsequent analysis much cleaner and more intuitive than working with raw NumPy arrays.

The following code block executes the data loading, handles the necessary structural transformations, renames the columns for brevity, and displays the initial six observations to confirm the data integrity and structure:

#load *iris* dataset
iris = datasets.load_iris()

#convert dataset to pandas DataFrame
df = pd.DataFrame(data = np.c_, iris],
columns = iris +)
df = pd.Categorical.from_codes(iris.target, iris.target_names)
df.columns =

#view first six rows of DataFrame
df.head()

s_length s_width p_length p_width target species
0 5.1 3.5 1.4 0.2 0.0 setosa
1 4.9 3.0 1.4 0.2 0.0 setosa
2 4.7 3.2 1.3 0.2 0.0 setosa
3 4.6 3.1 1.5 0.2 0.0 setosa
4 5.0 3.6 1.4 0.2 0.0 setosa

#find how many total observations are in dataset
len(df.index)

150

The processed dataset consists of 150 total observations, each representing a single flower measurement. Our core task is to train the LDA model to accurately classify the flower's species based exclusively on its physical dimensions.

The model will leverage the following four distinct features, acting as **predictor variables**:

Sepal length
Sepal width
Petal length
Petal width

These measurements will be utilized to predict the categorical outcome, or Species, which is divided into the following three mutually exclusive classes:

Iris setosa
Iris versicolor
Iris virginica

Step 3: Training the Linear Discriminant Analysis Model

With the data correctly partitioned and structured, we move to the central stage: instantiating and training the LDA model. The **scikit-learn** library streamlines this process significantly through the use of the `LinearDiscriminantAnalysis()` class, which adheres to the standard fit-predict API common across the library.

Before fitting the model, it is crucial to clearly delineate our input and output data. We define the feature matrix (X), which encompasses the four physical measurement columns, and the target vector (y), which holds the categorical species classification. Failure to separate these correctly will lead to training errors.

The snippet below demonstrates the assignment of variables and the subsequent fitting of the model using the `fit()` method. In this step, the model learns the optimal linear boundaries by analyzing the relationships within the entire Iris dataset:

#define predictor and response variables
X = df]
y = df

#Fit the LDA model
model = LinearDiscriminantAnalysis()
model.fit(X, y)

Step 4: Robust Performance Evaluation using Cross-Validation

After training the model, assessing its generalization capability on unseen data is mandatory to

ensure its reliability. We employ **cross-validation**, a robust statistical technique, to provide an unbiased estimate of the model's predictive accuracy. This method involves partitioning the data into multiple folds, training the model on a subset, and validating it on the remaining portion, repeating this process systematically.

Specifically, we utilize Repeated Stratified K-Fold Cross Validation. The "Stratified" element ensures that the proportion of each species class is maintained across all training and testing splits, which is vital for balanced classification evaluation. We configure the process to use 10 folds and repeat the entire procedure 3 times, yielding 30 total independent evaluation scores, thereby minimizing variance in the final accuracy estimate.

The evaluation results below calculate and display the mean accuracy score derived from all 30 cross-validation runs:

```
#Define method to evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

#evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print(np.mean(scores))
```

```
0.9777777777777779
```

The computed mean **accuracy** score of approximately **97.78%** is outstanding. This high score confirms that the LDA model is highly effective at distinguishing between the three distinct species categories based solely on the four provided physical measurements, demonstrating excellent generalization capabilities.

Using the Trained Model for New Predictions

While evaluation confirms model quality, the practical utility of a classification algorithm is demonstrated by its ability to accurately predict outcomes for new, previously unseen data points. The `predict()` method is used for this purpose, applying the learned linear boundaries to new input features to assign a most probable class membership.

To test this functionality, let us define a hypothetical new flower observation using a list of specific measurements. Suppose we encounter a flower with the following dimensions: Sepal Length = 5.0, Sepal Width = 3.0, Petal Length = 1.0, and Petal Width = 0.4. We pass these measurements to the trained model:

```
#define new observation
```

new =

#predict which class the new observation belongs to
model.predict()

array(, dtype='<U10')

The model efficiently processes these new inputs and, based on its established discriminant function, confidently predicts that this hypothetical specimen belongs to the species **setosa**, confirming the practical application of the trained classifier.

Step 5: Visualizing Class Separation with the LDA Plot

A significant advantage of LDA is its inherent capability for dimensionality reduction. It achieves this by finding the axes (linear discriminants) that maximize class separation. Since we have three classes, LDA can project the four original features onto a two-dimensional space (the maximum number of discriminants is C-1, where C is the number of classes), allowing for the creation of an informative **LDA plot**.

This visualization is essential as it provides direct insight into the effectiveness of the classification. We can visually inspect how distinct and tightly clustered the projected groups are. Well-separated clusters confirm that the linear boundaries derived by the model are robust and highly effective at distinguishing the underlying groups, thereby validating the statistical performance visually.

The code provided below utilizes the **Matplotlib** library to generate this visualization. It projects the 150 observations onto the first two principal discriminant axes, color-coding the points according to their true species classification:

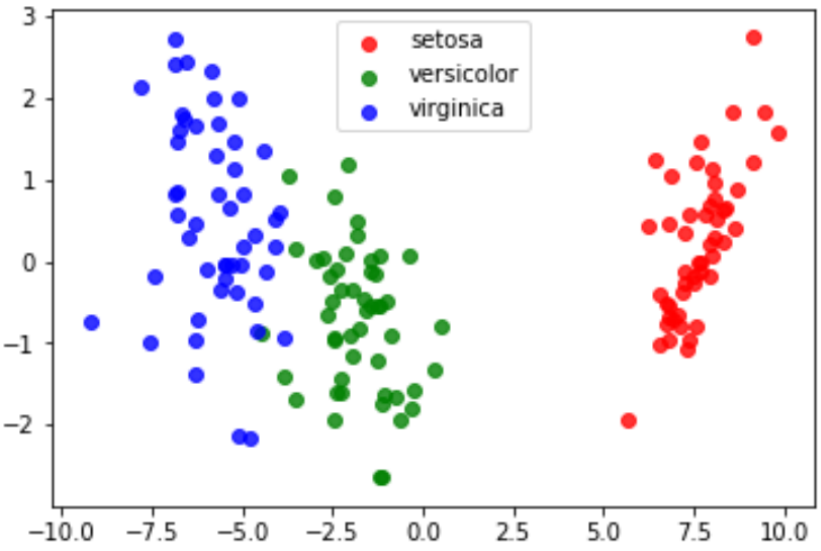
#define data to plot
X = iris.data
y = iris.target
model = LinearDiscriminantAnalysis()
data_plot = model.fit(X, y).transform(X)
target_names = iris.target_names

#create LDA plot
plt.figure()
colors =
lw = 2
for color, i, target_name in zip(colors, , target_names):
plt.scatter(data_plot, data_plot, alpha=.8, color=color,

```
label=target_name)

#add legend to plot
plt.legend(loc='best', shadow=False, scatterpoints=1)

#display LDA plot
plt.show()
```



The visualization provides compelling evidence of the model's success. As clearly demonstrated in the plot, the LDA transformation successfully separates the three species into distinct, non-overlapping clusters. This visual confirmation perfectly reinforces the high classification accuracy score (97.78%) obtained during the earlier cross-validation procedures, validating the robustness of the linear discriminants.

Conclusion and Further Resources

This tutorial successfully guided you through the entire methodological pipeline for implementing and evaluating Linear Discriminant Analysis within the Python environment, utilizing the efficient tools provided by the **scikit-learn** library. LDA remains a foundational, highly effective statistical method, particularly well-suited for classification tasks where features approximate a normal distribution and clear separation between classes is desired.

By mastering these steps--from initial library configuration and meticulous data preparation to sophisticated model fitting, robust cross-validation, and insightful data visualization--you are now fully equipped to apply LDA to a wide variety of your own real-world classification challenges, ensuring high performance and interpretability.

For comprehensive review and immediate execution, the complete, executable Python script utilized throughout this guide is readily available for download.

You can access the full Python code used in this tutorial here.