Learning Quadratic Discriminant Analysis with Python: A Step-by-Step Guide

Authored by

Mohammed looti

November 6, 2025

Quadratic Discriminant Analysis (QDA) is a sophisticated statistical approach utilized for classification tasks where the objective is to assign a response variable into one of two or more discrete categories based on a collection of predictor variables. QDA is exceptionally well-suited for scenarios where the optimal decision boundaries separating these classes are not straightforward lines but rather complex, curved, or quadratic forms.

The fundamental theoretical basis of QDA rests on the assumption that the observations belonging to each class are independently drawn from a Gaussian distribution. Crucially, unlike its simpler relative, Linear Discriminant Analysis (LDA), QDA permits each class to possess its own unique statistical characteristics, specifically assuming a distinct covariance matrix. This crucial allowance for differing variances across categories is what grants QDA the flexibility to model complex, non-linear separation boundaries between the classes.

Because QDA relaxes the strict requirement of equal covariance structures across all classes (a core assumption of LDA), it is often considered the non-linear extension of LDA. While LDA remains computationally efficient and performs adequately when the assumption of equal covariance holds true, QDA typically delivers superior classification accuracy in real-world applications where this assumption is frequently violated, although this precision comes at the cost of estimating a greater number of parameters.

This comprehensive tutorial provides a practical, step-by-step guide demonstrating the implementation and rigorous evaluation of a **Quadratic Discriminant Analysis** model. We will leverage the capabilities of the **Python** programming language and the industry-standard machine learning library, scikit-learn, ensuring a robust and reproducible workflow.

Step 1: Setting Up the Environment and Loading Libraries

The foundation of any successful machine learning endeavor is the meticulous preparation of the development environment and the installation of necessary tools. For this QDA implementation, we primarily rely on the robust scikit-learn (sklearn) library, which houses the core classification algorithm via the `QuadraticDiscriminantAnalysis` class. Additionally, essential libraries for data handling and numerical operations, namely Pandas and NumPy, must be imported.

Beyond the core data processing tools, we specifically import modules designed for rigorous model testing and validation. These include `train_test_split` for initial data partitioning and `RepeatedStratifiedKFold`, which is crucial for unbiased performance assessment using advanced cross-validation techniques. We also import the `datasets` module from `sklearn` to quickly access a well-known benchmark dataset for immediate demonstration.

Ensure that all these dependencies are correctly installed within your Python environment before executing the following code snippet, which loads all required packages into the current session.

```
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 QuadraticDiscriminantAnalysis
from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Step 2: Data Acquisition and Preparation

To effectively demonstrate the mechanics of classification algorithms like QDA, we utilize the Iris dataset, which stands as a universally recognized benchmark in machine learning. This dataset comprises 150 observations detailing physical measurements of iris flowers, categorized across three distinct species, making it perfectly suited for multi-class classification exercises.

While scikit-learn conveniently provides this data, it is initially structured as a NumPy array. For enhanced readability, easier manipulation, and clearer assignment of feature and target names, we convert this structure into a robust pandas DataFrame. The code below handles the loading, conversion, and precise labeling of all columns, ensuring the dataset is clean and organized for modeling.

Following the data loading process, it is standard practice to inspect the initial rows and confirm the total number of entries. This immediate verification confirms that we are working with 150 observations in total, 50 belonging to each species. Our primary modeling objective is to construct a **Quadratic Discriminant Analysis** classifier capable of accurately predicting the flower's species based exclusively on its four physical measurements.

```
#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

We must now formally define the two critical components required for the QDA model training process: the predictor variables and the response variable.

The **Predictor Variables** (Features) used to inform the classification decision are:

Sepal length (s_length)
Sepal width (s_width)
Petal length (p_length)
Petal width (p_width)

The **Response Variable** (Target) that the model will learn to predict is: *Species*, which encompasses the three potential classes:

setosa
versicolor
virginica

Step 3: Fitting the Quadratic Discriminant Analysis (QDA) Model

With our dataset now fully prepared and structured, the next crucial step involves partitioning the data into the independent variables (features, denoted as X) and the dependent variable (target, denoted as y). This clear separation is fundamental, as the classification model must learn the complex relationships embedded within X to accurately predict y.

We proceed by initializing the QDA classifier using the dedicated QuadraticDiscriminantAnalysis class from the **scikit-learn** library. A significant advantage of QDA, especially for initial modeling, is that it generally functions optimally using its default settings. Its performance primarily relies on the inherent statistical characteristics of the training data rather than extensive manual hyperparameter tuning.

The core training process is executed via the `.fit()` method. During this step, the QDA algorithm

estimates the statistical parameters for each class distribution--specifically calculating the mean vector and the unique class-specific covariance matrix for each species, using the provided feature data (X) and their corresponding species labels (y).

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

#Fit the QDA model
model = QuadraticDiscriminantAnalysis()
model.fit(X, y)

Step 4: Evaluating Model Performance using Cross-Validation

After the QDA model has been fitted to the training data, a critical step is the rigorous evaluation of its generalization ability. Relying on a simple accuracy score obtained directly from the training data can be misleading due to the risk of overfitting. To secure a stable and reliable estimate of how the model will perform on novel, unseen data, we implement a robust cross-validation methodology.

We have chosen the **Repeated Stratified K-Fold Cross-Validation** technique. This method enhances reliability by dividing the dataset into *k* folds (we use 10 folds) and ensuring that the proportion of each target class remains consistent across all folds (stratification). Furthermore, we repeat this entire K-Fold process multiple times (3 repeats) to minimize variance in the resulting performance metric, yielding a highly stable measure of model effectiveness.

We utilize the `cross_val_score` utility provided by scikit-learn, setting 'accuracy' as the primary scoring metric. The scores returned represent the accuracy achieved in each fold across all repetitions. The subsequent calculation of the mean of these scores provides the definitive overall expected performance of our **Quadratic Discriminant Analysis** classifier.

#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))

0.97333333333334

The outcome of the rigorous cross-validation process demonstrates that the QDA model achieved

an outstanding mean classification accuracy of **97.33%**. This exceptional result confirms the model's high effectiveness and its ability to accurately distinguish between the three iris species based solely on the four input physical measurements.

Step 5: Applying the Model to New Observations

The final operational stage of any classification pipeline is using the validated model for prediction on data it has never encountered before. Since our QDA model has proven its reliability through cross-validation, it is now ready to transition from evaluation to practical deployment.

To test its practical utility, we define a hypothetical new data point--a vector containing a set of sepal and petal measurements. We feed these values into our trained QDA model using the `.predict()` method. The model then applies the quadratic decision boundaries it calculated during the fitting process to determine the most probable species class for this new observation.

This step perfectly illustrates the practical power of the QDA classifier, yielding a precise and definitive classification result based solely on the input feature values.

#define new observation
new =

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

array(, dtype='<U10')

Based on the input measurements (Sepal Length: 5.0, Sepal Width: 3.0, Petal Length: 1.0, Petal Width: 0.4), the trained **Quadratic Discriminant Analysis** model confidently predicts that this new observation belongs to the species *setosa*. This successful classification confirms the model's readiness for real-world application.

Conclusion and Resources

This tutorial has provided a comprehensive and executable walkthrough of implementing Quadratic Discriminant Analysis in Python. We successfully covered all essential stages: initial environment setup, meticulous data preparation, robust model fitting, and rigorous performance evaluation using Repeated Stratified K-Fold cross-validation, resulting in a highly accurate predictive classifier.

QDA is a particularly powerful tool to select when domain knowledge suggests that the underlying classes exhibit distinct statistical properties, specifically unequal covariance matrix structures. By

accommodating these non-linear classification boundaries, QDA often provides superior discriminatory power compared to simpler linear methodologies, especially when the data features are assumed to be normally distributed.

To further study the complete Python code utilized throughout this guide, or to adapt this methodology for your unique classification problems, the full source code is available here.