How do you perform Linear Discriminant Analysis in Python step-by-step?

Authored by

stats writer

April 21, 2024

Linear Discriminant Analysis (LDA) is a statistical technique used for classifying data by reducing the dimensionality of the data while preserving the discriminatory information between classes. In Python, LDA can be performed in a few simple steps.

Step 1: Importing the necessary libraries
The first step is to import the necessary libraries such as numpy, pandas, and sklearn, which contain functions for performing LDA.

Step 2: Loading the data
The next step is to load the dataset into the Python environment. This can be done using the pandas library, which allows for easy manipulation and analysis of data.

Step 3: Pre-processing the data
Before performing LDA, it is important to pre-process the data by removing any missing values and encoding categorical variables if necessary. This step ensures that the data is in a suitable format for the LDA algorithm.

Step 4: Splitting the data into training and testing sets
The dataset is then split into two sets - a training set and a testing set. The training set is used to train the LDA model, while the testing set is used to evaluate the performance of the model.

Step 5: Standardizing the data
To avoid any bias in the LDA model, it is important to standardize the data by subtracting the mean and dividing by the standard deviation. This ensures that all variables are on the same scale.

Step 6: Fitting the LDA model
The LDA model can be fitted to the training data using the fit() function from the sklearn library. This step involves calculating the mean and covariance matrix for each class in the dataset.

Step 7: Calculating the discriminant scores
The discriminant scores are calculated using the transform() function, which takes in the training data as input. These scores represent the projection of the data onto the LDA axes.

Step 8: Evaluating the performance of the model
The performance of the LDA model can be evaluated by predicting the classes for the testing data using the predict() function and comparing them to the actual classes.

Step 9: Visualizing the results
To better understand the performance of the LDA model, the results can be visualized using various techniques such as scatter plots or confusion matrices.

In summary, performing Linear Discriminant Analysis in Python involves importing the necessary

libraries, loading and pre-processing the data, splitting the data into training and testing sets, standardizing the data, fitting the LDA model, calculating discriminant scores, evaluating the performance of the model, and visualizing the results.

Linear Discriminant Analysis in Python (Step-by-Step)

Linear discriminant analysis is a method you can use when you have a set of predictor variables and you'd like to classify a response variable into two or more classes.

This tutorial provides a step-by-step example of how to perform linear discriminant analysis in Python.

Step 1: Load Necessary Libraries

First, we'll load the necessary functions and libraries for this example:

```
from sklearn.model_selectionimport train_test_split
from sklearn.model_selectionimport RepeatedStratifiedKFold
from sklearn.model_selectionimport cross_val_score
from sklearn.discriminant_analysisimport LinearDiscriminantAnalysis
from sklearn import datasets
import matplotlib.pyplotas plt
```

```
import pandas as pd
import numpy as np
```

Step 2: Load the Data

For this example, we'll use the iris dataset from the sklearn library. The following code shows how to load this dataset and convert it to a pandas DataFrame to make it easy to work with:

```
#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 datasetlen(df.index)

150

We can see that the dataset contains 150 total observations.

For this example we'll build a linear discriminant analysis model to classify which species a given flower belongs to.

We'll use the following predictor variables in the model:

Sepal lengthSepal widthPetal lengthPetal width

And we'll use them to predict the response variable *Species*, which takes on the following three potential classes:

setosaversicolorvirginica

Step 3: Fit the LDA Model

Next, we'll fit the LDA model to our data using the LinearDiscriminantAnalsyis function from sklearn:

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

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

Step 4: Use the Model to Make Predictions

Once we've fit the model using our data, we can evaluate how well the model performed by using repeated stratified k-fold cross validation.

For this example, we'll use 10 folds and 3 repeats:

```
#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

We can see that the model performed a mean accuracy of 97.78%.

We can also use the model to predict which class a new flower belongs to, based on input values:

#define new observation
new =

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

array(, dtype='<U10')

We can see that the model predicts this new observation to belong to the species called *setosa*.

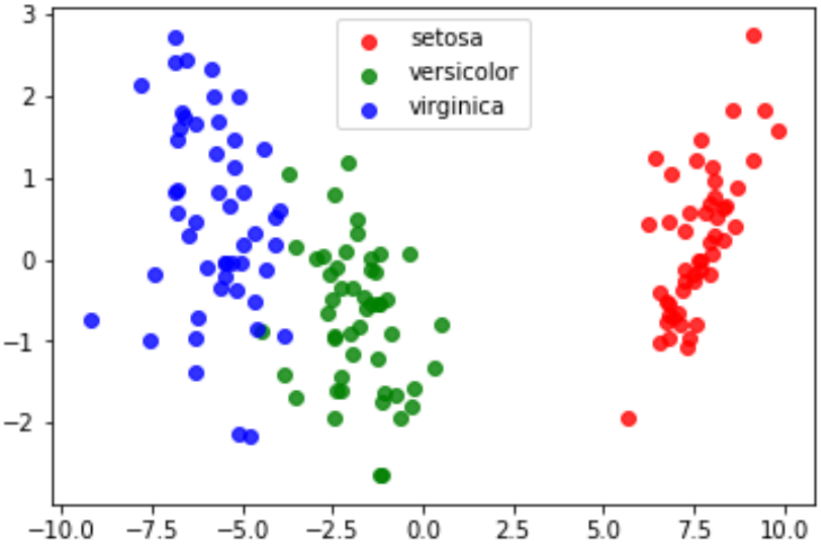
Step 5: Visualize the Results

Lastly, we can create an LDA plot to view the linear discriminants of the model and visualize how well it separated the three different species in our dataset:

```
#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()
```



You can find the complete Python code used in this tutorial here.