Linear Discriminant Analysis

Introduction

Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting (“curse of dimensionality”) and also reduce computational costs.

So, in a nutshell, often the goal of an LDA is to project a feature space (a dataset n-dimensional samples) onto a smaller subspace k (where k≤n−1) while maintaining the class-discriminatory information.  
In general, dimensionality reduction does not only help reducing computational costs for a given classification task, but it can also be helpful to avoid overfitting by minimizing the error in parameter estimation (“curse of dimensionality”).

Summarizing the LDA approach in 5 steps

Listed below are the 5 general steps for performing a linear discriminant analysis; we will explore them in more detail in the following sections.

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors (e1,e2,...,ed) and corresponding eigenvalues (λ1,λ2,...,λd) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d×k dimensional matrix W (where every column represents an eigenvector).
5. Use this d×k eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: Y=X×W (where X is a n×d-dimensional matrix representing the n samples, and y are the transformed n×k-dimensional samples in the new subspace).

Preparing the sample data set

About the Iris dataset

For the following tutorial, we will be working with the famous “Iris” dataset that has been deposited on the UCI machine learning repository  
(https://archive.ics.uci.edu/ml/datasets/Iris).

\*\*Reference:\*\* Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.

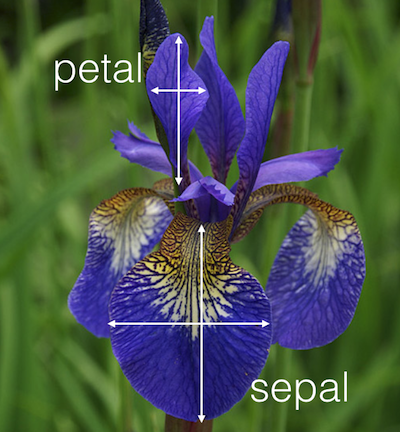
The iris dataset contains measurements for 150 iris flowers from three different species.

The three classes in the Iris dataset:

1. Iris-setosa (n=50)
2. Iris-versicolor (n=50)
3. Iris-virginica (n=50)

The four features of the Iris dataset:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm



LDA in 5 steps

After we went through several preparation steps, our data is finally ready for the actual LDA. In practice, LDA for dimensionality reduction would be just another preprocessing step for a typical machine learning or pattern classification task.

Step 1: Computing the d-dimensional mean vectors

np**.**set\_printoptions(precision**=**4)

mean\_vectors **=** []

**for** cl **in** range(1,4):

mean\_vectors**.**append(np**.**mean(X[y**==**cl], axis**=**0))

**print**('Mean Vector class %s: %s\n' **%**(cl, mean\_vectors[cl**-**1]))

Mean Vector class 1: [ 5.006 3.418 1.464 0.244]

Mean Vector class 2: [ 5.936 2.77 4.26 1.326]

Mean Vector class 3: [ 6.588 2.974 5.552 2.026]

Step 2: Computing the Scatter Matrices

Now, we will compute the two *4x4*-dimensional matrices: The within-class and the between-class scatter matrix.

2.1 Within-class scatter matrix SW

The **within-class scatter** matrix SW is computed by the following equation:

SW=∑i=1cSi

where  
Si=∑x∈Din(x−mi)(x−mi)T  
(scatter matrix for every class)

and mi is the mean vector  
mi=1ni∑x∈Dinxk

S\_W **=** np**.**zeros((4,4))

**for** cl,mv **in** zip(range(1,4), mean\_vectors):

class\_sc\_mat **=** np**.**zeros((4,4)) *# scatter matrix for every class*

**for** row **in** X[y **==** cl]:

row, mv **=** row**.**reshape(4,1), mv**.**reshape(4,1) *# make column vectors*

class\_sc\_mat **+=** (row**-**mv)**.**dot((row**-**mv)**.**T)

S\_W **+=** class\_sc\_mat *# sum class scatter matrices*

**print**('within-class Scatter Matrix:\n', S\_W)

within-class Scatter Matrix:

[[ 38.9562 13.683 24.614 5.6556]

[ 13.683 17.035 8.12 4.9132]

[ 24.614 8.12 27.22 6.2536]

[ 5.6556 4.9132 6.2536 6.1756]]

2.2 Between-class scatter matrix SB

The **between-class scatter** matrix SBSB is computed by the following equation:

SB=∑i=1cNi(mi−m)(mi−m)T

where  
m is the overall mean, and mi and Ni are the sample mean and sizes of the respective classes.

overall\_mean **=** np**.**mean(X, axis**=**0)

S\_B **=** np**.**zeros((4,4))

**for** i,mean\_vec **in** enumerate(mean\_vectors):

n **=** X[y**==**i**+**1,:]**.**shape[0]

mean\_vec **=** mean\_vec**.**reshape(4,1) *# make column vector*

overall\_mean **=** overall\_mean**.**reshape(4,1) *# make column vector*

S\_B **+=** n **\*** (mean\_vec **-** overall\_mean)**.**dot((mean\_vec **-** overall\_mean)**.**T)

**print**('between-class Scatter Matrix:\n', S\_B)

between-class Scatter Matrix:

[[ 63.2121 -19.534 165.1647 71.3631]

[ -19.534 10.9776 -56.0552 -22.4924]

[ 165.1647 -56.0552 436.6437 186.9081]

[ 71.3631 -22.4924 186.9081 80.6041]]

Step 3: Solving the generalized eigenvalue problem for the matrix SW−1SB

Next, we will solve the generalized eigenvalue problem for the matrix SW−1SB to obtain the linear discriminants.

eig\_vals, eig\_vecs **=** np**.**linalg**.**eig(np**.**linalg**.**inv(S\_W)**.**dot(S\_B))

**for** i **in** range(len(eig\_vals)):

eigvec\_sc **=** eig\_vecs[:,i]**.**reshape(4,1)

**print**('\nEigenvector {}: \n{}'**.**format(i**+**1, eigvec\_sc**.**real))

**print**('Eigenvalue {:}: {:.2e}'**.**format(i**+**1, eig\_vals[i]**.**real))

Eigenvector 1:

[[-0.2049]

[-0.3871]

[ 0.5465]

[ 0.7138]]

Eigenvalue 1: 3.23e+01

Eigenvector 2:

[[-0.009 ]

[-0.589 ]

[ 0.2543]

[-0.767 ]]

Eigenvalue 2: 2.78e-01

Eigenvector 3:

[[ 0.179 ]

[-0.3178]

[-0.3658]

[ 0.6011]]

Eigenvalue 3: -4.02e-17

Eigenvector 4:

[[ 0.179 ]

[-0.3178]

[-0.3658]

[ 0.6011]]

Eigenvalue 4: -4.02e-17

If we are performing the LDA for dimensionality reduction, the eigenvectors are important since they will form the new axes of our new feature subspace; the associated eigenvalues are of particular interest since they will tell us how “informative” the new “axes” are.

Let us briefly double-check our calculation and talk more about the eigenvalues in the next section.

Checking the eigenvector-eigenvalue calculation

A quick check that the eigenvector-eigenvalue calculation is correct and satisfy the equation:

Av=λv

where  
A=SW−1SB v=Eigenvector λ=Eigenvalue

**for** i **in** range(len(eig\_vals)):

eigv **=** eig\_vecs[:,i]**.**reshape(4,1)

np**.**testing**.**assert\_array\_almost\_equal(np**.**linalg**.**inv(S\_W)**.**dot(S\_B)**.**dot(eigv),

eig\_vals[i] **\*** eigv,

decimal**=**6, err\_msg**=**'', verbose**=**True)

**print**('ok')

ok

Step 4: Selecting linear discriminants for the new feature subspace

4.1. Sorting the eigenvectors by decreasing eigenvalues

So, in order to decide which eigenvector(s) we want to drop for our lower-dimensional subspace, we have to take a look at the corresponding eigenvalues of the eigenvectors. Roughly speaking, the eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data, and those are the ones we want to drop.

*# Make a list of (eigenvalue, eigenvector) tuples*

eig\_pairs **=** [(np**.**abs(eig\_vals[i]), eig\_vecs[:,i]) **for** i **in** range(len(eig\_vals))]

*# Sort the (eigenvalue, eigenvector) tuples from high to low*

eig\_pairs **=** sorted(eig\_pairs, key**=lambda** k: k[0], reverse**=**True)

*# Visually confirm that the list is correctly sorted by decreasing eigenvalues*

**print**('Eigenvalues in decreasing order:\n')

**for** i **in** eig\_pairs:

**print**(i[0])

Eigenvalues in decreasing order:

32.2719577997

0.27756686384

5.71450476746e-15

5.71450476746e-15

**Note**

If we take a look at the eigenvalues, we can already see that 2 eigenvalues are close to 0. The reason why these are close to 0 is not that they are not informative In LDA, the number of linear discriminants is at most c−1 where c is the number of class labels

Now, let’s express the “explained variance” as percentage:

**print**('Variance explained:\n')

eigv\_sum **=** sum(eig\_vals)

**for** i,j **in** enumerate(eig\_pairs):

**print**('eigenvalue {0:}: {1:.2%}'**.**format(i**+**1, (j[0]**/**eigv\_sum)**.**real))

Variance explained:

eigenvalue 1: 99.15%

eigenvalue 2: 0.85%

eigenvalue 3: 0.00%

eigenvalue 4: 0.00%

The first eigenpair is by far the most informative one, and we won’t loose much information if we would form a 1D-feature spaced based on this eigenpair.

4.2. Choosing *k* eigenvectors with the largest eigenvalues

After sorting the eigenpairs by decreasing eigenvalues, it is now time to construct our k×d-dimensional eigenvector matrix W (here 4×2: based on the 2 most informative eigenpairs) and thereby reducing the initial 4-dimensional feature space into a 2-dimensional feature subspace.

W **=** np**.**hstack((eig\_pairs[0][1]**.**reshape(4,1), eig\_pairs[1][1]**.**reshape(4,1)))

**print**('Matrix W:\n', W**.**real)

Matrix W:

[[-0.2049 -0.009 ]

[-0.3871 -0.589 ]

[ 0.5465 0.2543]

[ 0.7138 -0.767 ]]

Step 5: Transforming the samples onto the new subspace

In the last step, we use the 4×2-dimensional matrix W that we just computed to transform our samples onto the new subspace via the equation

Y=X×W.

(where X is a n×d-dimensional matrix representing the nn samples, and Y are the transformed n×k-dimensional samples in the new subspace).

X\_lda **=** X**.**dot(W)

**assert** X\_lda**.**shape **==** (150,2), "The matrix is not 150x2 dimensional."

from matplotlib import pyplot **as** plt

**def** **plot\_step\_lda**():

ax **=** plt**.**subplot(111)

**for** label,marker,color **in** zip(

range(1,4),('^', 's', 'o'),('blue', 'red', 'green')):

plt**.**scatter(x**=**X\_lda[:,0]**.**real[y **==** label],

y**=**X\_lda[:,1]**.**real[y **==** label],

marker**=**marker,

color**=**color,

alpha**=**0.5,

label**=**label\_dict[label]

)

plt**.**xlabel('LD1')

plt**.**ylabel('LD2')

leg **=** plt**.**legend(loc**=**'upper right', fancybox**=**True)

leg**.**get\_frame()**.**set\_alpha(0.5)

plt**.**title('LDA: Iris projection onto the first 2 linear discriminants')

*# hide axis ticks*

plt**.**tick\_params(axis**=**"both", which**=**"both", bottom**=**"off", top**=**"off",

labelbottom**=**"on", left**=**"off", right**=**"off", labelleft**=**"on")

*# remove axis spines*

ax**.**spines["top"]**.**set\_visible(False)

ax**.**spines["right"]**.**set\_visible(False)

ax**.**spines["bottom"]**.**set\_visible(False)

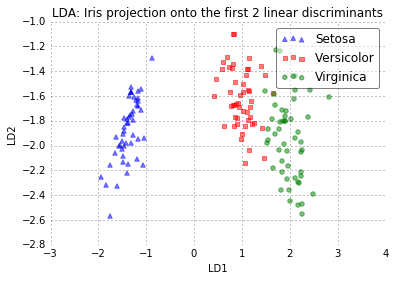
ax**.**spines["left"]**.**set\_visible(False)

plt**.**grid()

plt**.**tight\_layout

plt**.**show()

plot\_step\_lda()



The scatter plot above represents our new feature subspace that we constructed via LDA. We can see that the first linear discriminant “LD1” separates the classes quite nicely. However, the second discriminant, “LD2”, does not add much valuable information, which we’ve already concluded when we looked at the ranked eigenvalues is step 4.