ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 4, Issue 3, March 2017

Extraction Of Object In Multiviews Based On LDA Approach

[1] Geeravani,[2] Dr. P.Mohanaiah

[1] [2] NBKR Institute of Science and Technology

Abstract— 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 over fitting (―curse of dimensionality‖) and also reduce computational costs. Ronald A. Fisher formulated the Linear Discriminant in 1936, and it also has some practical uses as classifier. The original Linear discriminant was described for a 2-class problem, and it was then later generalized as ―multi-class Linear Discriminant Analysis‖ or ―Multiple Discriminant Analysis‖ by C. R. Rao in 1948 The general LDA approach is very similar to a Principal Component Analysis (for more information about the PCA, see the previous article, but in addition to finding the component axes that maximize the variance of our data (PCA), we are additionally interested in the axes that maximize the separation between multiple classes (LDA). 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 kk (where k≤n−1k≤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 over fitting by minimizing the error in parameter estimation (―curse of dimensionality‖).

I. INTRODUCTION

- Principal Component Analysis vs. Linear Discriminant Analysis
- What is a “good” feature subspace?
- Summarizing the LDA approach in 5 steps
- LDA in 5 steps

Step 1: Computing the d-dimensional mean vectors
Step 2: Computing the Scatter Matrices
2.1 Within-class scatter matrix SWSW
2.2 class-covariance matrices
2.3 Between-class scatter matrix SBSB
Step 3: Solving the generalized eigenvalue problem for the matrix S−1WSBSW−1SB
Checking the eigenvector-eigenvalue calculation
Step 4: Selecting linear discriminants for the new feature subspace
4.1. Sorting the eigenvectors by decreasing eigenvalues
4.2. Choosing k eigenvectors with the largest eigenvalues
Step 5: Transforming the samples onto the new subspace

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 over fitting (“curse of dimensionality”) and also reduce computational costs.

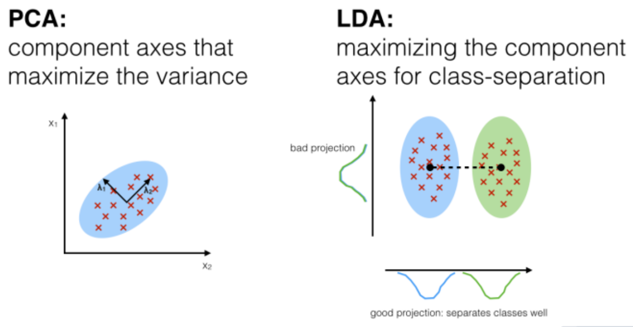
Ronald A. Fisher formulated the Linear Discriminant in 1936, and it also has some practical uses as classifier. The original Linear discriminant was described for a 2-class problem, and it was then later generalized as “multi-class Linear Discriminant Analysis” or “Multiple Discriminant Analysis” by C. R. Rao in 1948 The general LDA approach is very similar to a Principal Component Analysis (for more information about the PCA, see the previous article, but in addition to finding the component axes that maximize the variance of our data (PCA), we are additionally interested in the axes that maximize the separation between multiple classes (LDA).

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 kk (where k≤n−1k≤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 over fitting by minimizing the error in parameter estimation (“curse of dimensionality”).

Principal component analysis vs. Linear discriminant analysis

Both Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are linear transformation techniques that are commonly used for dimensionality reduction. PCA can be described as an “unsupervised” algorithm, since it “ignores” class labels and its goal is to find the directions (the so-called principal components) that maximize the variance in a dataset. In contrast to PCA, LDA is “supervised” and computes the directions (“linear discriminants”) that will represent the axes that that maximize the separation between multiple classes.

Although it might sound intuitive that LDA is superior to PCA for a multi-class classification task where the class labels are known, this might not always the case. For example, comparisons between classification accuracies for image recognition after using PCA or LDA show that PCA tends to outperform LDA if the number of samples per class is relatively. In practice, it is also not uncommon to use both LDA and PCA in combination: E.g., PCA for dimensionality reduction followed by an LDA.


PCA:
component axes that maximize the variance
LDA:
maximizing the component axes for class-separation
bad projection
good projection: separates classes well

What is a "good" feature subspace?

Let's assume that our goal is to reduce the dimensions of a dd-dimensional dataset by projecting it onto a(k)(k)-dimensional subspace (where k<dk<d). So, how do we know what size we should choose for kk (kk = the number of dimensions of the new feature subspace), and how do we know if we have a feature space that represents our data "well"? Later, we will compute eigenvectors (the components) from our data set and collect them in also-called scatter-matrices (i.e., the in-between-class scatter matrix and within-class scatter matrix). Each of these eigenvectors is associated with an eigenvalue, which tells us about the "length" or "magnitude" of the eigenvectors.

If we would observe that all eigenvalues have a similar magnitude, then this may be a good indicator that our data is already projected on a "good" feature space.

And in the other scenario, if some of the eigenvalues are much larger than others, we might be interested in keeping only those eigenvectors with the highest eigenvalues, since they contain more information about our data distribution. Vice versa, eigenvalues that are close to 0 are less informative and we might consider dropping those for constructing the new feature subspace.

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 dd-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 (ee1,ee2,...,eedee1,ee2,...,eed) and corresponding eigenvalues (λλ1,λλ2,...,λλdλλ1,λλ2,...,λλd) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose kk eigenvectors with the largest eigenvalues to form a k×dk×d dimensional matrix WWWW (where every column represents an eigenvector).
5. Use this k×dk×d eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the mathematical equation: YY=XX×WWYY=XX×WW (where XXXX is a n×dn×d-dimensional matrix representing the nn samples, and yyyy are the transformed n×kn×k-dimensional samples in the new subspace).

LDA Description of each 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

In this first step, we will start off with a simple computation of the mean vectors mmimmi, (i=1,2,3)(i=1,2,3) of the 3 different flower classes:

mmi=μωi(sepal length)μωi(sepal width)μωi(petal length)μωi(petal width), with i=1,2,3mmi=[μωi(sepal length)μωi(sepal width)μωi(petal length)μωi(petal width)],with i=1,2,3

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 4, Issue 3, March 2017

```
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 3: Solving the generalized eigenvalue problem for the matrix S−1WSBSW−1SB

Next, we will solve the generalized eigenvalue problem for the matrix S−1WSBSW−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
```

After this decomposition of our square matrix into eigenvectors and eigenvalues, let us briefly recapitulate how we can interpret those results. As we remember from our first linear algebra class in high school or college, both eigenvectors and eigenvalues are providing us with information about the distortion of a linear transformation: The eigenvectors are basically the direction of this distortion, and the eigenvalues are the scaling factor for the eigenvectors that describing the magnitude of the distortion.

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.
Step 4: Selecting linear discriminants for the new feature subspace

4.1. Sorting the eigenvectors by decreasing eigenvalues

Remember from the introduction that we are not only interested in merely projecting the data into a subspace that improves the class separability, but also reduces the dimensionality of our feature space, (where the eigenvectors will form the axes of this new feature subspace). However, the eigenvectors only define the directions of the new axis, since they have all the same unit length 1. 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. The common approach is to rank the eigenvectors from highest to lowest corresponding eigenvalue and choose the top kk eigenvectors.

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

eig_pairs= [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i inrange(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 ineig_pairs:

print(i[0])

Eigenvalues in decreasing order:

32.2719577997

0.27756686384

5.71450476746e-15

5.71450476746e-15
```

If we take a look at the eigenvalues, we can already see that 2 eigenvalues are close to 0 and conclude that the eigen pairs are less informative than the other two. Let's express the "explained variance" as percentage:

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

eigv_sum=sum(eig_vals)

fori,jinenumerate(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 Eigen pair is by far the most informative one, and we won't lose 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×dk×d-dimensional eigenvector matrix WWWW (here 4×24×2: 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 4×24×2-dimensional matrix WWWW that we just computed to transform our samples onto the new subspace via the equation

YY=XX×WWYY=XX×WW.

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

```
X_lda=X.dot(W)

assertX_lda.shape== (150,2), "The matrix is not 2x150 dimensional."

frommatplotlib import pyplotasplt

defplot_step_lda():

ax=plt.subplot(111)

forlabel,marker,colorinzip(

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]

            )
```