
INSAID

Is LDA a dimensionality reduction technique or a classifier algorithm?

Introduction

In my last post, I started a discussion about dimensionality reduction which the matter was the real impact over the results using principal component analysis (PCA) before perform a classification task (https://meigarom.github.io/blog/pca.html). In this post, I am going to continue discussing this subject, but now, talking about Linear Discriminant Analysis (LDA) algorithm. LDA is defined as a dimensionality reduction technique by authors, however some sources explain that LDA actually works as a linear classifier.

In order to understand better these definitions, I am proposing here a simple test: I am going to apply LDA over the same dataset twice, each time using LDA with a different role. Measuring accuracy from both approaches, we can have a clear picture about which role works best for LDA. Thus, let's start.

A little bit about LDA

The idea behind LDA is simple. Mathematically speaking, we need to find a new feature space to project the data in order to maximize classes separability. Well, obviously the first step is to find a way to measure this capacity of separation of each new feature space candidate. The distance between the projected means of each class could be one of the measures, however only this distance would not be a very good metric because it does not take the spread of data into account. In 1988, a statistician called Ronald Fisher proposed the following solution: Maximize the function that represents the difference between the means, normalized by a measure of the within-class variability. The Fisher's propose is basically to maximize the distance between the mean of each class and minimize the spreading within the class itself. Thus, we come up with two measures: the within-class and the between-class. However, this formulation is only possible if we assume that the dataset has a Normal distribution. This assumption might bring a disadvantage because if the distribution of your data is significantly non-Gaussian, the LDA might not perform very well. In the next posts, I will deep dive in the code of LDA algorithm showing its implementation, here I am concerned in to find an answer for the initial question. Is LDA is a linear classifier or a dimensionality reduction technique?.

Details about the experiment

The experiment proposed in this post consist of a classification task over a public dataset. Basically, the dataset contains 846 observations and 18 features from 3 different types of vehicles. The features correspond to physical measures of the shape of these vehicle like compactness, circularity and radius, for example. You can check the dataset here: https://archive.ics.uci.edu/ml/datasets/Statlog+(Vehicle+Silhouettes. The original dataset brings observation from 4 types of vehicles, I am going to pick up examples from only 3 classes in order to be able to plot the new feature space.

LDA as a classifier algorithm

As I mentioned in the beginning, I am going to apply LDA over the same dataset twice with different role each time. In the first approach, LDA will work as a classifier and posteriorly it will reduce the dimensionality of the dataset and a neural network will perform the classification task, the results of both approaches will be compared afterwards.

In R language, LDA as a classifier is straightforward as follows:

```
library( dplyr )
# Load dataset
data_raw = read.csv( "../dataset/vehicle.csv", stringsAsFactor = FALSE )
data = data_raw %>% filter( class == "bus" | class == "opel" | class == "van" )
# Scale dataset
maxs = apply( data[,1:18], 2, max )
mins = apply( data[,1:18], 2, min )
dataset = as.data.frame( scale( data[,1:18], center = mins, scale = maxs - mins ) )
dataset = cbind( dataset, "class" = data$class )
# Split dataset
index = sample( 1:nrow( dataset ), round( nrow( dataset )*0.6 ), replace = FALSE )
X_train = dataset[ index, ]
test = dataset[ -index, ]
```

First of all, I loaded the dataset and filter only rows from bus, open and van classes. Later, I scaled the dataset using standard technique and then I split the dataset in training and test set with 60% and 40% of examples of each, respectively. With the datasets ready, we are able to apply LDA over the training dataset.

```
# Model Discriminant Analysis
library( MASS )
model = lda( class ~ ., data = X_train )
# Ploting LDA Model
projected_data = as.matrix( X_train[, 1:18] ) %*% model$scaling
plot( projected_data, col = X_train[,19], pch = 19 )
```

The LDA is modeled using MASS R library, it brings a couple model parameters such as prior probabilities of groups, the group means and the coefficients of linear discriminant. The most important result here is the coefficients, they are values that describe the new feature space where the data will be project in. LDA reduces dimensionality from original number of feature to C—1 features, where C is the number of classes. In this case, we have 3 classes, therefore the new feature space will have only 2 features. The above picture is the plot of the new feature space with the only two features, we can see the new position of the data, there is a few points overlapping between the three classes, but in general, the dataset is pretty separable. After training phase, we need to measure the accuracy of the model obtained.

```
# Model Testing
X_test = test[, !( names( test ) %in% c( "class" ) ) ]
model.results = predict( model, X_test )
# Results - Confusion Matrix
library( caret )
t = table( model.results$class, test$class )
print( confusionMatrix( t ) )
## Confusion Matrix and Statistics
##
##
##        bus opel van
##   bus   87    5   0
##   opel   2   75   0
##   van    0    5  78
##
## Overall Statistics
##
```

```
##                Accuracy : 0.952381
##                  95% CI : (0.9182971, 0.9751557)
##     No Information Rate : 0.3531746
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.9285056
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: bus Class: opel Class: van
## Sensitivity           0.9775281   0.8823529  1.0000000
## Specificity           0.9693252   0.9880240  0.9712644
## Pos Pred Value        0.9456522   0.9740260  0.9397590
## Neg Pred Value        0.9875000   0.9428571  1.0000000
## Prevalence            0.3531746   0.3373016  0.3095238
## Detection Rate        0.3452381   0.2976190  0.3095238
## Detection Prevalence  0.3650794   0.3055556  0.3293651
## Balanced Accuracy     0.9734266   0.9351884  0.9856322
```

The last part of this first approach is to apply the test dataset in the model and measure its results. As we can see, LDA reached around 95% of accuracy as a classifier which is pretty good result. LDA basically projects the data in a new linear feature space, obviously the classifier will reach high accuracy if the data are linear separable. Now that we have in mind the accuracy of LDA working like classifier, let's check the second approach.

LDA as a dimensionality reduction algorithm

Linear Discriminant Analysis also works as a dimensionality reduction algorithm, it means that it reduces the number of dimension from original to C — 1 number of features where C is the number of classes. In this example, we have 3 classes and 18 features, LDA will reduce from 18 features to only 2 features. After reducing, neural network model will be applied to classification task. The procedure here is almost the same as the previous one. Take a look at it.

```
# New Dataset
new_X_train = as.matrix( X_train[,1:18] ) %*% model$scaling
new_X_train = as.data.frame( new_X_train )
new_X_train$class = X_train$class
```

I will use the coefficients of linear discriminant already defined in the previous training phase to project the training dataset into the new feature space, this new space will be the new training dataset.

```
# Multi-Class -> Transforming The Labels
new_X_train = cbind( new_X_train, opel = new_X_train$class == "opel" )
new_X_train = cbind( new_X_train, van = new_X_train$class == "van" )
new_X_train = cbind( new_X_train, bus = new_X_train$class == "bus" )
new_X_train = new_X_train[, !( names( new_X_train ) %in% c( "class" ) ) ]
```

Before to model, we need to transform the dataset to be able to use neural network model as a multi-class classification problem. The modification consist of to create new columns for each label with values “True” or “False”. For example, all example from “opel” will receive “True” value below the “opel” column and “False” in otherwise label columns. Hence, the dataset is ready for multi-class model.

```
# Model Neural Network
library( neuralnet )
n = names( new_X_train )
f = as.formula( "opel+van+bus ~ LD1+LD2" )
nn = neuralnet( f, new_X_train, hidden = 3, linear.output = FALSE, lifesign = "full",
                threshold = 0.02, stepmax = 1e6 )
## hidden: 3    thresh: 0.02    rep: 1/1    steps:    1000  min thresh: 0.7307363799
##                                                    2000  min thresh: 0.7307363799
##                                                    3000  min thresh: 0.7307363799
##                                                    4000  min thresh: 0.6448965006
##                                                    5000  min thresh: 0.4485495524
##                                                    6000  min thresh: 0.3650681285
##                                                    7000  min thresh: 0.2756491212
##                                                    8000  min thresh: 0.2137706548
##                                                    9000  min thresh: 0.1995287711
##                                                   10000  min thresh: 0.173313487
##                                                   11000  min thresh: 0.173313487
##                                                   12000  min thresh: 0.173313487
##                                                   13000  min thresh: 0.173313487
##                                                   14000  min thresh: 0.1420902136
##                                                   15000  min thresh: 0.08689029484
##                                                   16000  min thresh: 0.08255271673
##                                                   17000  min thresh: 0.0690402594
##                                                   18000  min thresh: 0.04721150014
##                                                   19000  min thresh: 0.02719626607
##                                                   19149  error: 11.92918 time: 4.03
secs
```

The neural network model starts very simple with 3 hidden layer only, square mean error threshold of 0.02 and 1e6 maximum epochs. The idea here is apply the neural network over the new dataset and to check if we can reach better results. The test dataset will be the same used in the previous approach.

```
# Testing
X_test = as.matrix( test[,1:18] ) %*% model$scaling
nn.results = compute( nn, X_test )
# Results
print( "... resulting ..." )
## [1] "... resulting ..."
idx = apply( nn.results$net.result, c(1), function( x ){ which( x == max( x ) ) } )
predictions = c( "opel", "van", "bus")[idx]
# Confusion Matrix
library( caret )
t = table( predictions, test$class )
print( confusionMatrix( t ) )
## Confusion Matrix and Statistics
##
##
```

```
## predictions bus opel van
##        bus   84    3   1
##        opel   4   81   3
##        van    1    1  74
##
## Overall Statistics
##
##                Accuracy : 0.9484127
##                  95% CI : (0.9133996, 0.9722494)
##     No Information Rate : 0.3531746
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.9224872
##  Mcnemar's Test P-Value : 0.7667396
##
## Statistics by Class:
##
##                      Class: bus Class: opel Class: van
## Sensitivity           0.9438202   0.9529412  0.9487179
## Specificity           0.9754601   0.9580838  0.9885057
## Pos Pred Value        0.9545455   0.9204545  0.9736842
## Neg Pred Value        0.9695122   0.9756098  0.9772727
## Prevalence            0.3531746   0.3373016  0.3095238
## Detection Rate        0.3333333   0.3214286  0.2936508
## Detection Prevalence  0.3492063   0.3492063  0.3015873
## Balanced Accuracy     0.9596402   0.9555125  0.9686118
```

The neural network model reached around 93% of accuracy being applied over the dataset with dimension reduced. This result is very very close to the result from first approach. It is a import clue about the best work mode for LDA.

Results from both approaches

Well, I guess we finished the experiment. The curious thing here is that both results are pretty close, in terms of accuracy, it indicates that LDA works greatly in both modes. Comparing difficult of implementation, the first approach is more straightforward than the second one, the LDA model provides the classification result itself while in the second one we have more code because LDA just reduces the dimensionality and neural network model is in charge of performing the classification task.

Conclusion

The motivation question to write this post was: Is LDA a dimensionality reduction technique or a classifier? In my point of view, based on results and efforts of implementation, the answers is that LDA works fine in both modes, as well in classifier mode as in dimensionality reduction mode, I will give you supportive argument for this conclusion.

First of all, the characteristics of the dataset that you are working will guide you about the decision of apply LDA as a classifier or a dimensionality reduction algorithm to perform a classification task. The main of Linear Discriminant Analysis is basically separate example of classes linearly moving them to a different feature space, therefore if your dataset is linear separable, only applying LDA as a classifier you will get great results. However, if the dataset is not linear separable the LDA will try to organize your dataset in another space as the maximum linearly separability as possible, but it still be examples overlapping between classes because of

non-linearly characteristic of data. In this case, you will need to use another classification model to deal with nonlinear data such as neural network with multiple hidden layers, neural network with radial basis function or SVM with nonlinear Kernels.

Summarizing, the decision of LDA modes will depend of the dataset that you have in hands, if it is linearly separable LDA as a classifier is a solution fast with great results, but if the characteristic of the dataset is nonlinear, LDA will be a extra tool to be applied over the dataset in order to try to “make things better” or facilitate the job of the posterior classifier.

I have seen some works on this line where authors combine PCA and LDA, both for dimensionality reduction. Firstly, PCA acts reducing the features by its variance, then LDA apply linear dimensionality reduction and lastly a classifier model is performed over this modified dataset. It is definitely a roadmap to be followed in order to improve the results in a classification tasks.

Thanks!

The complete code is available on my git hub repository as well as the dataset.