Comparison of R and Python for Discriminant Analysis

Tina Shi

Jan. 16, 2017

In the book Multivariate Statistical Inference and Applications by Alvin C. Rencher, the discriminant analysis is introduced as classification, and based on a vector y of variables measured on a sampling unit, we wish to classify the unit into one of the two populations. Here we assume $y_1|G_1 \sim N(\mu_1, \Sigma_1), y_2|G_2 \sim N(\mu_2, \Sigma_2)$, p_1 and p_2 are the prior probabilities that y will come from G_1 and G_2 respectively, where $p_2 = 1 - p_1$.

How do we classify the new observation y? When $\Sigma_1 = \Sigma_2$, the linear classification rule is defined to be: Assign y to G_1 if

$$(\mu_1 - \mu_2)^{'}\Sigma^{-1}y > \frac{1}{2}(\mu_1 - \mu_2)^{'}\Sigma^{-1}(\mu_1 + \mu_2) + ln(\frac{p_1}{p_2})$$

and to G_2 otherwise.

Here μ_1, μ_2, and Σ are all unknown, how do we find this line? The way presented in the text book is relatively simple, we can use $\bar{y}_1, \bar{y}_2, S_1, S_2$ to estimate μ_1, μ_2, and Σ. Then our linear classification rule (or linear discriminant rule) is to assign y to G_1 if

$$(\bar{y}_1 - \bar{y}_2)^{'}S_p l^{-1}y > \frac{1}{2}(\bar{y}_1 - \bar{y}_2)^{'}S_p l^{-1}((\bar{y}_1 + \bar{y}_2)^{'} + ln(\frac{p_1}{p_2})$$

and to G_2 otherwise, where $S_{pl} = \frac{(n_1-1)S_1+(n_2-1)S_2}{n_1+n_2-2}$.

If $\Sigma_1 \neq \Sigma_2$, then logarithm of the density ratio for the multivariate normal is

$$Q(y) = \frac{1}{2}ln(\frac{|\Sigma_1|}{|\Sigma_2|}) - \frac{1}{2}(\mu_1^{'}\Sigma_1^{-1}\mu_1 - \mu_2^{'}\Sigma_2^{-1}\mu_2) + (\mu_1^{'}\Sigma_1^{-1} - \mu_2^{'}\Sigma_2^{-1})y - \frac{1}{2}y^{'}(\Sigma_1^{-1} - \Sigma_2^{-1})y.$$

The optimal classification rule is: Assign y to G_1 if

$$Q(y) > ln(\frac{p_1}{p_2})$$

and to G_2 otherwise.

Again it uses the sample statistics to estimate the population parameters, $Q(y)$ is estimated by

$$Q(y) = \frac{1}{2}ln(\frac{|S_1|}{|S_2|}) - \frac{1}{2}(\bar{y_1}_1^{'}S_1^{-1}\bar{y_1} - \bar{y_2}^{'}S_2^{-1}\bar{y_2}) + (\bar{y_1}^{'}S_1^{-1} - \bar{y_2}^{'}S_2^{-1})y - \frac{1}{2}y^{'}(S_1^{-1} - S_2^{-1})y.$$

Here is the dataset coming from the book exercise:

group	y1	y2	y3	group	y1	y2	y3
1	8	60	58	2	6.2	49	30
1	8	156	68	2	5.6	31	23
1	8	90	37	2	5.8	42	22
1	6.1	44	27	2	5.7	42	14
1	7.4	207	31	2	6.2	40	23
1	7.4	120	32	2	6.4	49	18
1	8.4	65	43	2	5.8	31	17
1	8.1	237	45	2	6.4	31	19
1	8.3	57	60	2	5.4	62	26
1	7	94	43	2	5.4	42	16
1	8.5	86	40				
1	8.4	52	48				
1	7.9	146	52				

Assume $p_1 = p_2$, we will use this dataset to find the rule and apply this rule to the data itself and check the misclassification rate. The estimated rule is equivalent to the following: Assign y to G_1 if $(y-\bar{y_1})' S_{pl}^{-1}(y-\bar{y_1}) - (y-\bar{y_2})' S_{pl}^{-1}(y-\bar{y_2}) < 0$, and to G_2 otherwise.

```
library(MASS)
soil<- read.csv(file='soil.csv',sep=',', header=T)
# extract data for group 1
x1 = soil[soil$group==1, 2:4]
# extract data for group 2
x2 = soil[soil$group==2, 2:4]
# the number of observations in group 1
n1 = nrow(x1)
# the number of observations in group 2
n2 = nrow(x2)
s1 = cov(x1)
s2 = cov(x2)
# compute the pooled covariance matrix
sp = ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
# extract all the data without the group label for prediction
x = soil[, 2:4]
# create an empty list
l <- c()
# use a for loop to compute the value on the left side of the equation
# if it is less than 0, then it is assigned to group 1
for (i in 1:nrow(x)){
  D1 <- t(unlist(x[i,]-colMeans(x1)))%*%ginv(sp)%*%unlist(x[i,]-colMeans(x1))
  D2 <- t(unlist(x[i,]-colMeans(x2)))%*%ginv(sp)%*%unlist(x[i,]-colMeans(x2))
  l[i] <- D1-D2
}
# check how many values are less than 0
l<0

##  [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE

# print out the predicted value
l

##  [1] -16.275931 -24.339131 -13.067873  10.544917 -12.730700  -7.812484
```

```
##  [7] -16.908290 -24.652847 -19.496036  -4.985973 -18.414835 -17.325018
## [13] -18.973428   8.562002  17.136725  14.773590  17.647314  10.757299
## [19]   9.462373  16.612563  10.291634  16.536087  20.095512
```

We find that only the fourth observation is misclassified to group 2 since the first 13 observations belong to group 1 and the last 10 observations belong to group 2. We can also find the value $(y-\bar{y_1})^{'}S_{pl}^{-1}(y-\bar{y_1})-(y-\bar{y_2})^{'}S_{pl}^{-1}(y-\bar{y_2})$ for the fourth observation is . It is much larger than 0. The misclassification rate is $\frac{1}{23}=0.0435$.

But if we use the packages in R or Python to work out this problem, there are some differences in the results.

In R, the lda package uses Fisher's linear discriminant, which does not make some of the assumptions of LDA such as normally distributed or equal class covariance. It finds a linear combination of features that have means and variances for $w \cdot y$ that have means $w \cdot \mu_i$ and variances $w^T \Sigma_i w$ for $i = 1, 2$. This w is found by maximizing the ratio of the variance between the classes to the variance within the classes. In other words, it tries to find w such that it will make the items in the same group closer to each other while the items in different groups farther from each other. When the assumptions of LDA are satisfied, the equation is equivalent to LDA. The vector w is the normal to the discriminant hyperplane. In a two-dimensional problem, the line that best divides the two groups is perpendicular to w. Generally, the data points are projected onto w; then the threshold c that best separates the data is chosen from analysis of the one-dimensional distribution and we determine the rule to be: assign y to group 1 if $w \cdot y > c$. However, if projections of points from both classes exhibit approximately the same distributions, a good choice would be the hyperplane between projections of the two means, $w \cdot \mu_1$ and $w \cdot \mu_2$, which is $c = w \cdot \frac{1}{2}(\mu_1 + \mu_2)$. In our case, we simply set $c = 0$.

Here is the R code for running the lda package on the same dataset.

```
library(MASS)
soil<- read.csv(file='soil.csv',sep=',', header=T)
soil.lda <- lda(group~.,data=soil,prior=c(1,1)/2)
```

We can get the normalized matrix which transforms observations to discriminant functions by specify scaling at the result of lda in R. Here is the R code with the output:

```
# the coefficients of the lda function
soil.lda$scaling

##             LD1
## y1 -1.293393648
## y2 -0.007866473
## y3 -0.031516505

# compute the decision function based on the coefficients
t(as.matrix(x) %*% soil.lda$scaling)

##          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## LD1 -12.64709 -13.71744 -12.22124 -9.086772 -12.17648 -11.52362 -12.73104
##          [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## LD1 -13.75909 -13.07455 -11.14841 -12.93102 -12.78636 -13.00517 -9.349993
##         [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
## LD1 -8.211745 -8.525438 -8.143967 -9.058579 -9.230474 -8.281324 -9.120394
##         [,22]     [,23]
## LD1 -8.291476 -7.818982
```

All the values are negative, are they all classified as group 1? No. Look at the result of prediction and the confusion matrix:

```
predict(soil.lda, soil)$class
```

```
##  [1] 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
## Levels: 1 2
```

```
ct <- table(soil$group, predict(soil.lda)$class)
ct
```

```
##
##      1  2
##   1 12  1
##   2  0 10
```

The result of the confusion matrix also shows that only the fourth observation has been misclassified to group 2 when it belongs to group 1. The misclassification rate is also $\frac{1}{23} = 0.0435$. How does R compute the classification rule? Since the scaling result has been normalized in order to make the within groups covariance matrix spherical, we also need to normalize the input data to get the right value. Here is the R code and output for computing the decision rule based on normalized data:

```
means = (colMeans(x1)+colMeans(x2))/2
scaled_decision = scale(as.matrix(x), center = means, scale = FALSE) %*% soil.lda$scaling
t(scaled_decision)
```

```
##          [,1]      [,2]     [,3]     [,4]      [,5]      [,6]      [,7]
## LD1 -2.160542 -3.230889 -1.73469 1.399781 -1.689932 -1.037065 -2.244484
##          [,8]      [,9]     [,10]    [,11]     [,12]    [,13]   [,14]
## LD1 -3.272533 -2.587994 -0.661861 -2.44447 -2.299803 -2.51862 1.13656
##        [,15]    [,16]    [,17]    [,18]    [,19]    [,20]    [,21]
## LD1 2.274808 1.961115 2.342586 1.427974 1.256079 2.205228 1.366159
##        [,22]    [,23]
## LD1 2.195077 2.667571
```

Clearly, the fourth observation has been classified to group 2 instead of group 1. This is what the prediction result gives to you in R when you call x after applying predict to the lda:

```
t(predict(soil.lda, soil)$x)
```

```
##             1         2        3        4         5         6         7
## LD1 -2.160542 -3.230889 -1.73469 1.399781 -1.689932 -1.037065 -2.244484
##             8         9        10       11        12       13      14
## LD1 -3.272533 -2.587994 -0.661861 -2.44447 -2.299803 -2.51862 1.13656
##           15       16       17       18       19       20       21
## LD1 2.274808 1.961115 2.342586 1.427974 1.256079 2.205228 1.366159
##           22       23
## LD1 2.195077 2.667571
```

In Python, the scaling result is the same as the one in R, however the values reported for classification are quite different. The python 3 code for finding the line, which utilizes sklearn package, is shown below:

```
# import the packages needed
import numpy as np
import pandas as pd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# import the data
soil = pd.read_csv("soil.csv",sep=',',header=0)
# specify the x and y
```

```
x = soil[['y1','y2','y3']]
y = soil['group']
# fit the model
clf = LinearDiscriminantAnalysis(priors=[0.5,0.5])
clf.fit(x,y)
print(clf.predict(x))
# print the scaling result
print(clf.scalings_)
print(clf.decision_function(x))
```

The prediction and scaling results are the same as before:
prediction result:[1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2];
scaling result: array([[-1.29339365],[-0.00786647],[-0.0315165]]).

However, the decision function result is different:

```
[ -8.13796536 -12.16956556  -6.53393643   5.27245852
 -6.36534997  -3.90624219  -8.45414487 -12.32642339
 -9.74801817  -2.49298637  -9.20741753  -8.66250878
 -9.48671413   4.2810012    8.5683624    7.38679498
  8.82365703   5.37864944   4.73118641   8.30628133
  5.14581719   8.26804346  10.04775596]
```

What leads to this difference? The document shows that decision_function(x) predicts confidence scores for samples, and the confidence score here is the signed distance of that sample to the hyperplane. How does Python compute the confidence score? The Python reports the intercept and coefficients of the decision function, we just need to use the regression formula $intercept + X^T coefficients$ to get the confidence scores. Here shows the calculation in Python 3:

```
print((np.dot(x, clf.coef_.T)+clf.intercept_).transpose())
```

```
[ -8.13796536 -12.16956556  -6.53393643   5.27245852
 -6.36534997  -3.90624219  -8.45414487 -12.32642339
 -9.74801817  -2.49298637  -9.20741753  -8.66250878
 -9.48671413   4.2810012    8.5683624    7.38679498
  8.82365703   5.37864944   4.73118641   8.30628133
  5.14581719   8.26804346  10.04775596]
```

For this simple case, the results for three situations are the same even though the decision function are computed differently. But it might be very different for larger datasets and more complex situations.