HW11 Report

인대영 (2022311154)

2022-05-26

1. We define a function 'my_lda' which takes predictor variables (x) and response variable (y) as input and performs linear discriminant analysis. Assuming equal priors and equal misclassification costs, we calculate the linear discriminant functions (LDFs) of each population as

$$\hat{d}_i(\underset{\sim}{x}) = \underset{\sim}{\bar{x}}_i' S_p^{-1} \underset{\sim}{x} - \frac{1}{2} \underset{\sim}{\bar{x}}_i' S_p^{-1} \underset{\sim}{\bar{x}}_i + \log P_i \quad (i = 1, 2, \cdots, g)$$

Each observation is classified to the population which has the highest corresponding LDF value. We also save the coefficients of the LDFs as well as the accuracy using APER method. The code is shown below:

In [2]:
```
#1. Write a Python code to calculae the linear discriminant function (LDF) for several classes.
#   Your code should be able to predict the Y class based on the input X0 values.
#X1~X3: Species 1~3
#Assume equal priors, equal misclassification costs
#Classify to the class with largest LDF value

def my_lda(x,y):
    var = x.columns
    x=pd.DataFrame(pd.DataFrame.reset_index(x.T, drop=True)).T
    X1=x.iloc[:50]
    X2=x.iloc[50:100]
    X3=x.iloc[100:]

    #Calculate sample means and covariances
    n1, n2, n3 = y.value_counts()
    n = n1 + n2 + n3
    p=1/3 #Assume equal priors
    X1_mean=pd.DataFrame(X1.mean())
    X2_mean=pd.DataFrame(X2.mean())
    X3_mean=pd.DataFrame(X3.mean())
    S1=X1.cov()
    S2=X2.cov()
    S3=X3.cov()
    Sp = 1/(n1+n2+n3-3)*((n1-1)*S1+(n2-1)*S2+(n3-1)*S3)

    #Dataframe for classification
    classified_lda = pd.DataFrame(columns = ['Classified into TYPE'])

    #Calculate LDFs
    #d: slopes
    #m: constants
    d1 = X1_mean.T.dot(np.linalg.inv(Sp))
    d2 = X2_mean.T.dot(np.linalg.inv(Sp))
    d3 = X3_mean.T.dot(np.linalg.inv(Sp))
    m1 = -1/2*X1_mean.T.dot(np.linalg.inv(Sp)).dot(X1_mean).iloc[0,0]+np.log(p)
    m2 = -1/2*X2_mean.T.dot(np.linalg.inv(Sp)).dot(X2_mean).iloc[0,0]+np.log(p)
    m3 = -1/2*X3_mean.T.dot(np.linalg.inv(Sp)).dot(X3_mean).iloc[0,0]+np.log(p)

    #Coefficients of LDFs
    coeff1 = np.append(m1,d1)
    coeff2 = np.append(m2,d2)
    coeff3 = np.append(m3,d3)
    coeff1 = pd.DataFrame(coeff1, index = np.append('Constant',var), columns = ['Coefficients'])
    coeff2 = pd.DataFrame(coeff2, index = np.append('Constant',var), columns = ['Coefficients'])
    coeff3 = pd.DataFrame(coeff3, index = np.append('Constant',var), columns = ['Coefficients'])

    for i in range(len(x)):
        x0=pd.Series.reset_index(x.iloc[i,:], drop=True)
        #Linear discriminant functions for population i (i=1,2,...,g)
        d1_val = X1_mean.T.dot(np.linalg.inv(Sp)).dot(x0)[0]-1/2*X1_mean.T.dot(np.linalg.inv(Sp)).dot(X1_mean).iloc[0,0]+np.log(
        d2_val = X2_mean.T.dot(np.linalg.inv(Sp)).dot(x0)[0]-1/2*X2_mean.T.dot(np.linalg.inv(Sp)).dot(X2_mean).iloc[0,0]+np.log(
        d3_val = X3_mean.T.dot(np.linalg.inv(Sp)).dot(x0)[0]-1/2*X3_mean.T.dot(np.linalg.inv(Sp)).dot(X3_mean).iloc[0,0]+np.log(
        max_ind = np.argmax([d1_val,d2_val,d3_val])
        classified_lda.loc[i,:] = max_ind+1 #save classification

    #Classification result
    result=pd.concat([y,classified_lda], axis=1)

    #Calculate accuracy of LDA (APER method)
    classified_lda.columns = ['Species']
    error_lda = classified_lda==pd.DataFrame(y)
    correct, incorrect = error_lda.value_counts()
    accuracy_lda = correct/(correct+incorrect)

    return coeff1, coeff2, coeff3, result, accuracy_lda
```

2. We define a function 'my_qda' which takes predictor variables (X) and response variable (y) as input and performs quadratic discriminant analysis. Assuming equal priors and equal misclassification costs, we calculate the quadratic discriminant functions (QDFs) of each population as

$$\hat{d}_i^Q(\underline{x}) = -\frac{1}{2}\log|S_i| - \frac{1}{2}(\underline{x}-\underline{\bar{x}}_i)' S_i^{-1}(\underline{x}-\underline{\bar{x}}_i) + \log p_i \quad (i=1,2,\cdots,g)$$

Each observation is classified to the population which has the highest corresponding QDF value. We also save the classification result of a new observation as well as the accuracy using APER method. The code is shown below:

In [3]:
```
#2. Write a Python code to calculate the quadratic discriminant function (QDF) for several classes.
#   Your code should be able to predict the Y class based on the input X0 values.
#Assume equal priors, equal misclassification costs
#Classify to the class with largest QDF value

def my_qda(x,y,x_new):
    var = x.columns
    x=pd.DataFrame(pd.DataFrame.reset_index(x.T, drop=True)).T
    X1=x.iloc[:50]
    X2=x.iloc[50:100]
    X3=x.iloc[100:]
    X1=pd.DataFrame(pd.DataFrame.reset_index(X1.T, drop=True)).T
    X2=pd.DataFrame(pd.DataFrame.reset_index(X2.T, drop=True)).T
    X3=pd.DataFrame(pd.DataFrame.reset_index(X3.T, drop=True)).T

    #Calculate sample means and covariances
    n1, n2, n3 = y.value_counts()
    n = n1 + n2 + n3
    p=1/3 #Assume equal priors
    X1_mean=pd.DataFrame(X1.mean())
    X2_mean=pd.DataFrame(X2.mean())
    X3_mean=pd.DataFrame(X3.mean())
    S1=X1.cov()
    S2=X2.cov()
    S3=X3.cov()

    #Dataframe for classification and quadratic discriminant scores
    classified_qda = pd.DataFrame(columns = ['Classified into TYPE'])
    scores = pd.DataFrame(columns = ['QD score'])

    #Calculate QDFs
    for i in range(len(x)):
        x0=pd.DataFrame(x.iloc[i,:])
        x0.columns = [0]
        #quadratic discriminant functions for population i (i=1,2,...,g)
        d1_val = -1/2*np.log(np.linalg.det(S1))-1/2*((x0-X1_mean).T.dot(np.linalg.inv(S1))).dot(x0-X1_mean).iloc[0,0]+np.log(p)
        d2_val = -1/2*np.log(np.linalg.det(S2))-1/2*((x0-X2_mean).T.dot(np.linalg.inv(S2))).dot(x0-X2_mean).iloc[0,0]+np.log(p)
        d3_val = -1/2*np.log(np.linalg.det(S3))-1/2*((x0-X3_mean).T.dot(np.linalg.inv(S3))).dot(x0-X3_mean).iloc[0,0]+np.log(p)
        max_ind = np.argmax([d1_val,d2_val,d3_val])
        classified_qda.loc[i,:] = max_ind+1 #save classification
        scores.loc[i,:] = np.round(max([d1_val,d2_val,d3_val]),4) #save scores

    #Classification result
    result=pd.concat([y,classified_qda,scores], axis=1)

    #Classification for new observation
    result_new = pd.DataFrame(index=['x_new'],columns = ['Classified into TYPE','QD score'])
    d1_new = -1/2*np.log(np.linalg.det(S1))-1/2*((x_new-X1_mean).T.dot(np.linalg.inv(S1))).dot(x_new-X1_mean).iloc[0,0]+np.log(p
    d2_new = -1/2*np.log(np.linalg.det(S2))-1/2*((x_new-X2_mean).T.dot(np.linalg.inv(S2))).dot(x_new-X2_mean).iloc[0,0]+np.log(p
    d3_new = -1/2*np.log(np.linalg.det(S3))-1/2*((x_new-X3_mean).T.dot(np.linalg.inv(S3))).dot(x_new-X3_mean).iloc[0,0]+np.log(p
    max_new = np.argmax([d1_new,d2_new,d3_new])
    result_new.loc[:,:]=[max_new+1,np.round(max([d1_new,d2_new,d3_new]),4)] #save classification and score

    #Calculate accuracy of QDA (APER method)
    classified_qda.columns = ['Species']
    error_qda = classified_qda==pd.DataFrame(y)
    correct, incorrect = error_qda.value_counts()
    accuracy_qda = correct/(correct+incorrect)

    return result, result_new, accuracy_qda
```

3. We implement a function 'loo' to perform the 'leave-one-out' method to calculate the accuracy of the LDA and QDA models. Since we are 'holding-out' one observation before developing a classification function, we we a for-loop to iterate through each observation. Additionally, we divide cases for removing an observation from either $\pi 1$, $\pi 2$, or $\pi 3$ using an if-else statement. After calculating the LDFs and QDFs and classifying each observation accordingly, we count the number of correctly (incorrectly) predicted observations. Finally, we calculate the accuracies for LDA and QDA respectively. The code is shown on the next page.

```
def loo(x,y):
    var = x.columns
    x = pd.DataFrame.reset_index(x.T,drop=True).T
    p = 1/3
    classified_ldf = pd.DataFrame(columns = ['Species'])
    classified_qdf = pd.DataFrame(columns = ['Species'])

    for i in range(len(x)):
        n1, n2, n3 = y.value_counts()
        if i<50:
            n1=n1-1
            n = n1 + n2 + n3
            x_temp = x.drop(i)
            x_temp = pd.DataFrame.reset_index(x_temp,drop=True)
            X1 = x_temp.iloc[:49]
            X1 = pd.DataFrame.reset_index(X1,drop=True)
            X2 = x_temp.iloc[49:99]
            X2 = pd.DataFrame.reset_index(X2,drop=True)
            X3 = x_temp.iloc[99:]
            X3 = pd.DataFrame.reset_index(X3,drop=True)
        elif i<100:
            n2=n2-1
            n = n1 + n2 + n3
            x_temp = x.drop(i)
            x_temp = pd.DataFrame.reset_index(x_temp,drop=True)
            X1 = x_temp.iloc[:50]
            X1 = pd.DataFrame.reset_index(X1,drop=True)
            X2 = x_temp.iloc[50:99]
            X2 = pd.DataFrame.reset_index(X2,drop=True)
            X3 = x_temp.iloc[99:]
            X3 = pd.DataFrame.reset_index(X3,drop=True)
        elif i<150:
            n3=n3-1
            n = n1 + n2 + n3
            x_temp = x.drop(i)
            x_temp = pd.DataFrame.reset_index(x_temp,drop=True)
            X1 = x_temp.iloc[:50]
            X1 = pd.DataFrame.reset_index(X1,drop=True)
            X2 = x_temp.iloc[50:100]
            X2 = pd.DataFrame.reset_index(X2,drop=True)
            X3 = x_temp.iloc[100:]
            X3 = pd.DataFrame.reset_index(X3,drop=True)
        X1_mean=pd.DataFrame(X1.mean())
        X2_mean=pd.DataFrame(X2.mean())
        X3_mean=pd.DataFrame(X3.mean())
        S1=X1.cov()
        S2=X2.cov()
        S3=X3.cov()
        Sp = 1/(n1+n2+n3-3)*((n1-1)*S1+(n2-1)*S2+(n3-1)*S3)

        #Calculate LDFs for multiple classes
        x0=pd.Series.reset_index(x.iloc[i,:], drop=True)
        d1_val = X1_mean.T.dot(np.linalg.inv(Sp)).dot(x0)[0]-1/2*X1_mean.T.dot(np.linalg.inv(Sp)).dot(X1_mean).iloc[0,0]+np.log(
        d2_val = X2_mean.T.dot(np.linalg.inv(Sp)).dot(x0)[0]-1/2*X2_mean.T.dot(np.linalg.inv(Sp)).dot(X2_mean).iloc[0,0]+np.log(
        d3_val = X3_mean.T.dot(np.linalg.inv(Sp)).dot(x0)[0]-1/2*X3_mean.T.dot(np.linalg.inv(Sp)).dot(X3_mean).iloc[0,0]+np.log(
        max_ind_ldf = np.argmax([d1_val,d2_val,d3_val])
        classified_ldf.loc[i,:] = max_ind_ldf+1 #save classification

        #Calculate QDFs for multiple classes
        x0=pd.DataFrame(x.iloc[i,:])
        x0.columns = [0]
        d1_val = -1/2*np.log(np.linalg.det(S1))-1/2*((x0-X1_mean).T.dot(np.linalg.inv(S1))).dot(x0-X1_mean).iloc[0,0]+np.log(p)
        d2_val = -1/2*np.log(np.linalg.det(S2))-1/2*((x0-X2_mean).T.dot(np.linalg.inv(S2))).dot(x0-X2_mean).iloc[0,0]+np.log(p)
        d3_val = -1/2*np.log(np.linalg.det(S3))-1/2*((x0-X3_mean).T.dot(np.linalg.inv(S3))).dot(x0-X3_mean).iloc[0,0]+np.log(p)
        max_ind_qdf = np.argmax([d1_val,d2_val,d3_val])
        classified_qdf.loc[i,:] = max_ind_qdf+1 #save classification

    #Calculate accuracy of LDA (LOO method)
    error_lda = classified_ldf==pd.DataFrame(y)
    correct, incorrect = error_lda.value_counts()
    acc_lda_loo = correct/(correct+incorrect)

    #Calculate accuracy of QDA (LOO method)
    error_qda = classified_qdf==pd.DataFrame(y)
    correct, incorrect = error_qda.value_counts()
    acc_qda_loo = correct/(correct+incorrect)

    return acc_lda_loo, acc_qda_loo
```

4. (a) After saving Fisher's Iris data and dropping the 'Species' column, we save x1~x4 characteristics as 'x_iris' and the species as 'y_iris'. We use 'statsmodels.stats.multivariate' module to perform one-way analysis of covariance. Under H_0: equal covariance, we compute the chi-square test statistic and pvalue. Since pvalue $= 3.3520 \times 10^{-20} < 0.05$, we reject H_0. Thus, we should perform QDA on this data because covariances are not equal.

```
In [8]: test = mv.test_cov_oneway([cov1,cov2,cov3],[len(X1_test),len(X2_test),len(X3_test)])
        print("Chi-Square Test statistic:",test.statistic_chi2, " , Pr > ChiSq:",test.pvalue_chi2)

        #Since p-value is small enough, we reject H0 => QDA

        Chi-Square Test statistic: 140.94304992349774  , Pr > ChiSq: 3.352034178317213e-20
```

(b) We calculate the quadratic discriminant scores of the data and classify the new observation $X_0 = [5.0, 3.5, 1.75, 0.21)'$ using the code in #2. The corresponding result is shown below:

```
In [10]: #Classifications and respective quadratic discriminant scores of iris data
         result_qda
```

Out[10]:

	Species	Classified into TYPE	QD score
0	1	1	5.2105
1	1	1	4.3945
2	1	1	4.7929
3	1	1	4.582
4	1	1	5.0542
...	...	...	...
145	3	3	1.0922
146	3	3	1.3611
147	3	3	2.809
148	3	3	1.394
149	3	3	2.0194

150 rows × 3 columns

```
In [11]: #Classification and quadratic discriminant scores of new observation
         result_new
```

Out[11]:

	Classified into TYPE	QD score
x_new	1	3.491

(c) We calculate the LDFs and output their coefficients using the code in #1.

```
In [13]: coeff1
```

Out[13]:

	Coefficients
Constant	-86.308470
Sepal length(x1)	23.544167
Sepal width(x2)	23.587870
Petal length(x3)	-16.430639
Petal width(x4)	-17.398411

```
In [14]: coeff2
```

Out[14]:

	Coefficients
Constant	-72.852607
Sepal length(x1)	15.698209
Sepal width(x2)	7.072510
Petal length(x3)	5.211451
Petal width(x4)	6.434229

```
In [15]: coeff3
```

Out[15]:

	Coefficients
Constant	-104.368320
Sepal length(x1)	12.445849
Sepal width(x2)	3.685280
Petal length(x3)	12.766545
Petal width(x4)	21.079113

We compare our result with the LDFs obtained using 'sklearn.discriminant_analysis' module. We can see that the coefficients are slightly different.

In [17]:
```
#Compare with Python package
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(priors = [1/3,1/3,1/3], solver = 'eigen') # Equal prior
lda.fit(x_iris,y_iris)
```

Out[17]:
```
LinearDiscriminantAnalysis(priors=[0.3333333333333333, 0.3333333333333333,
                                   0.3333333333333333],
                           solver='eigen')
```

In [18]:
```
#Intercepts of LDFs
lda.intercept_
```

Out[18]:
```
array([ -88.04744666,  -74.31697465, -106.47586504])
```

In [19]:
```
#Coefficients of LDFs
lda.coef_
```

Out[19]:
```
array([[ 24.02465992,  24.06925561, -16.76595819, -17.75348039],
       [ 16.01858069,   7.21684677,   5.31780708,   6.56554   ],
       [ 12.69984591,   3.7604894 ,  13.02708671,  21.50929899]])
```

(d) We use the function defined in #3 to calculate the leave-one-out (LOO) error rates for LDA and QDA respectively. We previously used the functions defined in #1 and #2 to calculate the APER. Their respective values are shown below.

In [20]:
```
#(d) Calculate and compare the APER and the leave-one-out error rates for linear discriminant analysis (LDA) and quadratic discri
#    using your code in #1,2,3. (Assume equal prior and equal misclassification cost.)
#error=1-accuracy
#Leave-one-out (LOO)
acc_lda_loo, acc_qda_loo = loo(x_iris,y_iris)
```

In [21]:
```
#LDA accuracy (LOO)
acc_lda_loo
```

Out[21]:
```
0.98
```

In [22]:
```
#QDA accuracy (LOO)
acc_qda_loo
```

Out[22]:
```
0.9733333333333334
```

In [23]:
```
#Apparent error rates (APER)
#LDA accuracy (APER)
acc_lda_aper
```

Out[23]:
```
0.98
```

In [24]:
```
#QDA accuracy (APER)
acc_qda_aper
```

Out[24]:
```
0.98
```

5. $-\frac{1}{2}(x-\mu_1)'\Sigma^{-1}(x-\mu_1)+\frac{1}{2}(x-\mu_2)'\Sigma^{-1}(x-\mu_2)$

$=-\frac{1}{2}\left(x'\Sigma^{-1}x-2\mu_1'\Sigma^{-1}x+\mu_1'\Sigma^{-1}\mu_1-x'\Sigma^{-1}x+2\mu_2'\Sigma^{-1}x-\mu_2'\Sigma^{-1}\mu_2\right)$ $(\because \mu_1'\Sigma^{-1}x : \text{scalar})$

$=-\frac{1}{2}\left(-2(\mu_1-\mu_2)'\Sigma^{-1}x+\mu_1'\Sigma^{-1}\mu_1-\mu_2'\Sigma^{-1}\mu_2\right)$

$=(\mu_1-\mu_2)'\Sigma^{-1}x-\frac{1}{2}(\mu_1-\mu_2)'\Sigma^{-1}(\mu_1+\mu_2)$