HW10 Report

인대영(2022311154)

2022-05-19

1. We define a function 'lda' to calculate the linear discriminant function for binary class and predict the Y class based on the input X_0 values. We use the following

$$\hat{R}_1 : \ \hat{\underset{\sim}{a}}' \underset{\sim}{X} - \frac{1}{2}\left(\hat{\underset{\sim}{a}}' \bar{X}_1 + \hat{\underset{\sim}{a}}' \bar{X}_2\right) \geq \log\left[\frac{P_2\, C(1|2)}{P_1\, C(2|1)}\right]$$

where LHS is the linear discriminant function (LDF).

Depending on whether LHS is greater or smaller than RHS, we classify it as either the first type (X_1) or the second type (X_2). In our function, we assume equal misclassification cost, i.e, $C(1|2) = C(2|1)$. The function is able to handle different prior probabilities, taking them as the input variables 'p1' and 'p2', respectively. The function returns the coefficients of the LDF, the classification result along with the posterior probabilities, and the values of the LDF indicated in the function as variables 'coeff', 'result', and 'ldf', respectively. The code is shown below.

In [3]:
```python
#LDA: Linear Discriminant Analysis
#X1: domestic, X2: wild
#priors: p1, p2
def lda(x,y,p1,p2):
    var = x.columns
    x = pd.DataFrame.reset_index(x.T,drop=True).T
    X1 = x.iloc[14:]
    X1 = pd.DataFrame.reset_index(X1,drop=True)
    X2 = x.iloc[:14]
    X2 = pd.DataFrame.reset_index(X2,drop=True)

    n1, n2 = y.value_counts()
    n = n1 + n2
    X1_mean = mean_vector(X1)
    X2_mean = mean_vector(X2)
    S1 = covariance_matrix(X1)
    S2 = covariance_matrix(X2)
    Sp = (n1-1)/(n1+n2-2)*S1+(n2-1)/(n1+n2-2)*S2

    #Calculate LDF for binary class
    a = (X1_mean-X2_mean).dot(np.linalg.inv(Sp))
    Y1_mean = a.dot(X1_mean.T).iloc[0,0]
    Y2_mean = a.dot(X2_mean.T).iloc[0,0]
    m = 1/2*(Y1_mean + Y2_mean)
    coeff = np.append(m,a)
    coeff = pd.DataFrame(coeff, index = np.append('Constant',var), columns = ['Coefficients'])

    #Predict Y class
    ldf = a.dot(x.T)-m
    ldf.index=['ldf']
    log = np.log(p2/p1) #assume equal misclassification cost
    ldf_bool = ldf > log
    classified = classifier(ldf_bool,'DOMESTIC','WILD')

    #Calculate posterior probabilities
    X1_mean_rep = pd.concat([X1_mean]*len(x))
    X1_mean_rep = pd.DataFrame.reset_index(X1_mean_rep,drop=True)
    f1=np.exp(-1/2*(x-X1_mean_rep).dot(np.linalg.inv(Sp)).dot((x-X1_mean_rep).T))
    X2_mean_rep = pd.concat([X2_mean]*len(x))
    X2_mean_rep = pd.DataFrame.reset_index(X2_mean_rep,drop=True)
    f2=np.exp(-1/2*(x-X2_mean_rep).dot(np.linalg.inv(Sp)).dot((x-X2_mean_rep).T))
    post1 = p1*f1/(p1*f1+p2*f2)
    post2 = p2*f2/(p1*f1+p2*f2)

    #Classification result
    result=pd.concat([y,classified.T,pd.DataFrame(np.round(np.diag(post1),4)),pd.DataFrame(np.round(np.diag(post2),4))],axis=1)
    result.columns=['From TYPE','Classified into Type','DOMESTIC','WILD']

    return coeff, result, ldf
```

2. We implement a function 'loo' to perform the 'leave-one-out' method to calculate the accuracy of the LDA model. The code for calculating the LDF is quite similar to the code in #1. However, since we are 'holding-out' one observation before developing a classification function, we use a for-loop to iterate through each observation. Additionally, we divide cases for removing an observation of type 1 (X1) or type 2 (X2) using an if-else statement. After calculating the LDF and classifying each observation, we count the number of correctly (incorrectly) predicted observations. Finally, we calculate the accuracy and return its value. The code is shown below.

```python
def loo(x,y,p1,p2):
    var = x.columns
    x = pd.DataFrame.reset_index(x.T,drop=True).T
    classify_list=[]

    for i in range(len(x)):
        n1, n2 = y.value_counts()
        if i<14:
            n1=n1-1
            n=n1+n2
            x_temp = x.drop(i)
            x_temp = pd.DataFrame.reset_index(x_temp,drop=True)
            X1 = x_temp.iloc[13:]
            X1 = pd.DataFrame.reset_index(X1,drop=True)
            X2 = x_temp.iloc[:13]
            X2 = pd.DataFrame.reset_index(X2,drop=True)
        else:
            n2=n2-1
            n=n1+n2
            x_temp = x.drop(i)
            x_temp = pd.DataFrame.reset_index(x_temp,drop=True)
            X1 = x_temp.iloc[14:]
            X1 = pd.DataFrame.reset_index(X1,drop=True)
            X2 = x_temp.iloc[:14]
            X2 = pd.DataFrame.reset_index(X2,drop=True)

        X1_mean = mean_vector(X1)
        X2_mean = mean_vector(X2)
        S1 = covariance_matrix(X1)
        S2 = covariance_matrix(X2)
        Sp = (n1-1)/(n1+n2-2)*S1+(n2-1)/(n1+n2-2)*S2

        #Calculate LDF for binary class
        a = (X1_mean-X2_mean).dot(np.linalg.inv(Sp))
        Y1_mean = a.dot(X1_mean.T).iloc[0,0]
        Y2_mean = a.dot(X2_mean.T).iloc[0,0]
        m = 1/2*(Y1_mean + Y2_mean)
        coeff = np.append(m,a)
        coeff = pd.DataFrame(coeff, index = np.append('Constant',var), columns = ['Coefficients'])

        #Predict Y class
        ldf = a.dot(x.iloc[i,:])[0]-m
        log = np.log(p2/p1) #assume equal misclassification cost
        ldf_bool = ldf > log
        if ldf_bool>0:
            classified = 'DOMESTIC'
        else:
            classified = 'WILD'
        classify_list = np.append(classify_list, classified)

    classify_list = pd.DataFrame(classify_list, columns=['TYPE'])
    #calculate accuracy
    error = classify_list==pd.DataFrame(y)
    correct, incorrect = error.value_counts()
    accuracy = correct/(correct+incorrect)

    return accuracy
```

3. We read the Turkey data and pre-process it by removing 'ID' and 'TYPE' columns. Also, we only select male data.

(a) We apply the function 'lda' defined in the code in #1. We obtain the classified types as well as their respective posterior probabilities. From our result, the misclassified turkeys are ID = 'K766' (index = 0) and ID = 'L750' (index = 24).

In [10]:
```
#(a) Which turkeys in this data set were misclassified by the discriminant rule when the rule was applied to the training data?
#(b) What are the posterior probabilities for both domestic and wild classifcations for those turkeys that were misclassified in
result1
```

Out[10]:

	From TYPE	Classified into Type	DOMESTIC	WILD
0	WILD	DOMESTIC	0.6620	0.3380
1	WILD	WILD	0.0009	0.9991
2	WILD	WILD	0.0011	0.9989
3	WILD	WILD	0.0058	0.9942
4	WILD	WILD	0.0000	1.0000
5	WILD	WILD	0.0033	0.9967
6	WILD	WILD	0.0171	0.9829
7	WILD	WILD	0.0002	0.9998
8	WILD	WILD	0.0000	1.0000
9	WILD	WILD	0.0013	0.9987
10	WILD	WILD	0.0919	0.9081

(b) The posterior probabilities are also printed out from our result in #3(a).

	DOMESTIC	WILD
'K766'	0.6620	0.3380
'L750'	0.2645	0.7355

(c) Applying the 'lda' function also returns the values of the linear discriminant function as mentioned before.

In [11]:
```
#(c) Determine the value of each of the linear discriminant function for turkeys whose IDs are B710 and L674.
#How do you classify these two turkeys?
ldf1
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	23	24	25	26	
ldf	0.266544	-7.386383	-7.173841	-5.557687	-14.480549	-6.120478	-4.45412	-8.763461	-11.819885	-7.024255	...	4.986669	-1.428094	11.039951	9.077645	3.3

1 rows × 33 columns

In [12]:
```
#LDF values for 'B710' and 'L674'
ldf1.iloc[0,0], ldf1.iloc[0,14]
```

Out[12]: (0.2665441199626031, 8.85941578165395)

The LDF values for turkeys 'B710' and 'L674' are 0.2665 and 8.8594, respectively.

(d) We apply the code in #2 to obtain the 'leave-one-out' accuracy of the LDA model which turns out to be accuracy = 84.85%

In [13]:
```
#(d) Calculate the 'leave-one-out' accuracy of the LDA model.
acc = loo(x,y,0.6,0.4)
acc
```

Out[13]: 0.8484848484848485