HW3 Report

인대영 (2022311154)
2022-09-26

1. We first write a series of code in order to read the data for checking, "harris.dat". We set the first column as the response variable 'y' and the rest of the columns as predictor variables 'x'. The coefficients for each X variable is estimated using the gradient descent algorithm. A for loop was used to update the weights for each iteration and the calculated error was printed for every 10,000 epochs as shown below. The learning rate and number of epochs are set as 0.00004 and 10,000,000, respectively. The initial weights were selected randomly from unit [-1, 1].

Read Data

```
In [2]: harris = pd.read_csv("harris.dat", sep = " ", header=None)
```

```
In [3]: harris
```

Out[3]:

	0	1	2	3	4
0	3900	12	0.0	1	0
1	4020	10	44.0	7	0
2	4290	12	5.0	30	0
3	4380	8	6.2	7	0
4	4380	8	7.5	6	0
...	...	...	...	...	...
88	6600	15	215.5	16	1
89	6840	15	41.5	7	1
90	6900	12	175.0	10	1
91	6900	15	132.0	24	1
92	8100	16	54.5	33	1

93 rows × 5 columns

```
In [4]: x_col = harris.columns.delete(0)
        x = harris[x_col]
        x = x.T.reset_index(drop=True).T
        y = harris[0]
```

Gradient Descent Algorithm

```
In [6]: #initial weights
        np.random.seed(0)
        w = pd.Series(np.random.uniform(low=-1, high=1, size=4)) #weights
        b = np.random.uniform(low=-1, high=1) #constant
        w
```

```
Out[6]: 0    0.097627
        1    0.430379
        2    0.205527
        3    0.089766
        dtype: float64
```

```
In [*]: #Gradient Descent
        alpha=0.00004
        for i in range(1000000):
            y_hat = x.dot(w) + b
            error = np.abs(y_hat-y).mean()

            w[0] = w[0] - alpha*(((y_hat-y)*x[0]).mean())
            w[1] = w[1] - alpha*(((y_hat-y)*x[1]).mean())
            w[2] = w[2] - alpha*(((y_hat-y)*x[2]).mean())
            w[3] = w[3] - alpha*(((y_hat-y)*x[3]).mean())
            b = b - alpha*((y_hat-y).mean())
            if i%10000==0:
                print(i, np.round(error,4))
```

```
0 5372.304
10000 600.8853
20000 594.287
30000 588.0187
40000 582.0563
50000 576.3774
60000 571.1226
70000 566.4574
80000 562.0319
90000 557.8043
100000 553.7865
```

The estimated coefficients were also acquired using OLS via the 'statsmodels' Python package. The coefficients from each method are compared in the output file as shown below. Unfortunately, the estimated coefficients using gradient descent were not very accurate probably due to inadequate choice of learning rate and number of epochs and perhaps computational limitations as well. Methods to improve the estimations include reducing the learning rate or increasing the number of epochs which require higher computational cost.

HW3_output_descent.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
Coefficients by Gradient Descent Method
---------------
Constant:  2253.282
Beta1:  179.468
Beta2:  1.858
Beta3:  29.011
Beta4:  631.722

Coefficients by Statsmodels
---------------
Constant:  3526.422
Beta1:  90.02
Beta2:  1.269
Beta3:  23.406
Beta4:  722.461
```

2. We first prompt the user for the names of the training and test data files as well as whether to perform regression or classification. We define a function 'regression' to perform linear regression as in assignment #1. If regression is chosen, additional prompts are executed and an output file is written. The related code is shown below.

Regression

In [18]:
```
def regression(y,x):
    n = len(y)
    ones = pd.DataFrame(np.ones(n),dtype=np.int8)
    z = pd.concat([ones, x], axis=1)
    z = z.T.reset_index(drop=True).T #reset columns

    b_hat = pd.DataFrame((np.linalg.inv(z.T.dot(z))).dot(z.T).dot(y))
    b_hat = round(b_hat,3)

    y_hat = z.dot(b_hat)

    return b_hat, y_hat
```

In [19]:
```
if method == '1':
    #Prompt
    import_name=input("Please enter the name of the data file: ")
    encode_sep=int(input("Please select the separator used in the data file (1 = whitespace or 2 = comm
    separator={encode_sep ==1 : " "}.get(True, ",")
    res_pos =int(input("Please enter the position of the response variable column (select from 1 to p):
    header=input("Does the data file include a column header? (y/n) : ")
    export_name=input("Please enter the name for the file to be exported (e.g. result.txt) : ")

    #Multiple linear regression
    if(header=="y"):
        data=pd.read_csv(import_name, sep=separator)
    else:
        data=pd.read_csv(import_name, sep=separator, header=None)

    Y = data.iloc[:,res_pos-1] #response variable

    Z_col = data.columns.delete(res_pos-1) #columns of predictor variables
    Z=data[Z_col] #predictor variables
    ones=pd.DataFrame(np.ones(len(data),dtype=np.int8))
    Z=pd.concat([ones,Z],axis=1)
    Z = Z.T.reset_index(drop=True).T #reset columns#predictor variables with constant term

    B_hat=pd.DataFrame((np.linalg.inv(Z.T.dot(Z))).dot(Z.T).dot(Y))
    B_hat=round(B_hat,3)

    Y_hat = Z.dot(B_hat)
    Y_hat=round(Y_hat,3)

    #Calculate R^2
    SSE = sum((Y-Y_hat[0])**2)
    SST = sum((Y-Y.mean())**2)
    R_square = round(1-SSE/SST,4)

    #Calculate MSE
    n = len(data)
    p = len(Z.columns)
    MSE = round(SSE/(n-p),4)

    f = open(export_name,'w')
    text2 = "Coefficients\n------------\nConstant: "+str(B_hat.iloc[0,0])+"\n"
    for i in range(1,len(Z.columns)):
        text2 += "Beta"+str(i)+": "+str(B_hat.iloc[i,0])+"\n"
    text2 += "\nID, Actual values, Fitted values\n------------------------------\n"
    for j in range(5):
        text2 += str(j+1) + ", " + str(Y[j]) + ", " + str(Y_hat.iloc[j,0]) + "\n"
    text2 += "(continue)\n\nModel Summary\n------------\nR-square = " + str(R_square) + "\n"
    text2 += "MSE = " + str(MSE)
    f.write(text2)
    f.close()
```

In case 'classification' is chosen, we define a function 'lda' which performs linear discriminant analysis and returns the newly classified data. Assuming equal priors and equal misclassification costs, we calculate the linear discriminant functions (LDFs) of each population as

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

Each observation is classified to the population which has the highest corresponding LDF value.

Classification

In [20]:
```
def lda(x_lda, y_lda):
    X1=x_lda[y_lda==1]
    X2=x_lda[y_lda==2]
    X3=x_lda[y_lda==3]
    X4=x_lda[y_lda==4]

    n1, n2, n3, n4 = y_lda.value_counts(sort=False)
    n=n1+n2+n3+n4
    p=1/4 #Assume equal priors

    X1_mean = pd.DataFrame(X1.mean())
    X2_mean = pd.DataFrame(X2.mean())
    X3_mean = pd.DataFrame(X3.mean())
    X4_mean = pd.DataFrame(X4.mean())

    S1=X1.cov()
    S2=X2.cov()
    S3=X3.cov()
    S4=X4.cov()
    Sp = 1/(n1+n2+n3+n4-4)*((n1-1)*S1+(n2-1)*S2+(n3-1)*S3+(n4-1)*S4)

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

    for i in range(len(x_lda)):
        x0=pd.Series.reset_index(x_lda.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(
        d4_val = X4_mean.T.dot(np.linalg.inv(Sp)).dot(x0)[0]-1/2*X4_mean.T.dot(np.linalg.inv(Sp)).dot(X4_mean).iloc[0,0]+np.log(
        max_ind = np.argmax([d1_val,d2_val,d3_val,d4_val])
        classified_lda.loc[i,:] = max_ind+1 #save classification

    return classified_lda
```

We perform LDA on 'veh.dat' and 'vehtest.dat' and the confusion matrices and accuracy values are calculated using the code below. A portion is shown due to the length of the code.

In [27]:
```
if method == '2':
    #Classification using LDA
    train_result = lda(x_lda_train, y_lda_train)
    test_result = lda(x_lda_test, y_lda_test)

    #Confusion matrix
    #Train data
    conf_arr_train=np.array([[0,0,0,0],
                             [0,0,0,0],
                             [0,0,0,0],
                             [0,0,0,0]])

    for i in range(len(y_lda_train)):
        if y_lda_train[i]==1:
            if train_result.iloc[i,0]==1:
                conf_arr_train[0,0]+=1
            elif train_result.iloc[i,0]==2:
                conf_arr_train[0,1]+=1
            elif train_result.iloc[i,0]==3:
                conf_arr_train[0,2]+=1
            elif train_result.iloc[i,0]==4:
                conf_arr_train[0,3]+=1
        elif y_lda_train[i]==2:
            if train_result.iloc[i,0]==1:
                conf_arr_train[1,0]+=1
            elif train_result.iloc[i,0]==2:
                conf_arr_train[1,1]+=1
            elif train_result.iloc[i,0]==3:
                conf_arr_train[1,2]+=1
            elif train_result.iloc[i,0]==4:
                conf_arr_train[1,3]+=1
        elif y_lda_train[i]==3:
            if train_result.iloc[i,0]==1:
                conf_arr_train[2,0]+=1
            elif train_result.iloc[i,0]==2:
                conf_arr_train[2,1]+=1
            elif train_result.iloc[i,0]==3:
                conf_arr_train[2,2]+=1
            elif train_result.iloc[i,0]==4:
                conf_arr_train[2,3]+=1
        elif y_lda_train[i]==4:
            if train_result.iloc[i,0]==1:
                conf_arr_train[3,0]+=1
            elif train_result.iloc[i,0]==2:
                conf_arr_train[3,1]+=1
            elif train_result.iloc[i,0]==3:
                conf_arr_train[3,2]+=1
            elif train_result.iloc[i,0]==4:
                conf_arr_train[3,3]+=1

    #Test data
    conf_arr_test=np.array([[0,0,0,0],
                            [0,0,0,0],
                            [0,0,0,0],
                            [0,0,0,0]])
```

HW3_output_LDA.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

Confusion Matrix (Training)

		Predicted Class			
		1	2	3	4
Actual	0	76	23	2	4
Class	1	30	76	1	3
	2	2	1	106	1
	3	1	0	0	99

Model Summary (Training)

Overall accuracy = 0.84

Confusion Matrix (Test)

		Predicted Class			
		1	2	3	4
Actual	0	59	24	1	2
Class	1	32	45	3	5
	2	0	1	84	1
	3	0	2	1	76

Model Summary (Test)

Overall accuracy = 0.786

As shown above, the corresponding confusion matrix and accuracy is calculated for train and test data, respectively. The accuracy for train and test data are 0.84 and 0.786, respectively. Notice that the accuracy for train data is higher, which is natural since the LDA model is formed from the training data itself.