HW5 Report

인대영 (2022311154)
2022-10-10

1. We first prompt the user to enter the data sets. We then define a function 'ovr' which implements one-vs-rest method. The 'Logit' function from the 'statsmodels' package is utilized for fitting the binary logistic regression classifier. Once all the models for each class are fitted, the function returns each model as a list. We included the constant as well.

One-vs-rest

In [8]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

In [9]:
```
def ovr(x_train,y_train):
    n=len(y_train.unique()) #number of classes
    models=[]

    for i in range(1,n+1):
        classes=list(range(1,n+1)) #[1,2,3,4]
        classes.remove(i)

        y_train.replace() #replace "rest" with 0
        y_current=y_train.replace(classes,0).replace(i,1) #set value to 1
        model_current = sm.Logit(y_current,x_train).fit() #fit binary logistic regression model
        models.append(model_current)

    return models #return models of each class as a list
```

2. Next we fit the train data 'veh.dat' using 'ovr' function. A summary of the model for class 1 is shown below.

In [10]:
```
#Fit the train data from veh.dat
models_list=ovr(x_train,y_train) #save list of fitted models
```

```
Optimization terminated successfully.
         Current function value: 0.360581
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.327391
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.042271
         Iterations 13
Optimization terminated successfully.
         Current function value: 0.027253
         Iterations 16
```

In [11]:
```
#Summary of the first model (class 1)
models_list[0].summary()
```

Out[11]:

Logit Regression Results

Dep. Variable:	18	**No. Observations:**	425
Model:	Logit	**Df Residuals:**	406
Method:	MLE	**Df Model:**	18
Date:	Fri, 07 Oct 2022	**Pseudo R-squ.:**	0.3550
Time:	22:11:44	**Log-Likelihood:**	-153.25
converged:	True	**LL-Null:**	-237.61
Covariance Type:	nonrobust	**LLR p-value:**	1.615e-26

	coef	std err	z	P>|z|	[0.025	0.975]
const	63.3740	40.875	1.550	0.121	-16.740	143.488
0	-0.2447	0.046	-5.368	0.000	-0.334	-0.155
1	0.7207	0.188	3.827	0.000	0.352	1.090
2	-0.0163	0.035	-0.460	0.646	-0.086	0.053
3	0.1138	0.031	3.690	0.000	0.053	0.174
4	-0.3534	0.089	-3.969	0.000	-0.528	-0.179
5	0.0537	0.126	0.426	0.670	-0.193	0.301
6	0.3649	0.205	1.779	0.075	-0.037	0.767
7	0.3689	0.270	1.369	0.171	-0.159	0.897
8	-0.2934	0.510	-0.575	0.565	-1.293	0.706
9	-0.1785	0.062	-2.866	0.004	-0.301	-0.056
10	-0.0988	0.039	-2.518	0.012	-0.176	-0.022
11	-0.0338	0.028	-1.211	0.226	-0.088	0.021
12	-0.0409	0.016	-2.568	0.010	-0.072	-0.010
13	-0.3359	0.092	-3.643	0.000	-0.517	-0.155
14	0.0063	0.032	0.196	0.844	-0.057	0.069
15	0.0418	0.021	1.984	0.047	0.001	0.083
16	0.0654	0.118	0.555	0.579	-0.165	0.296
17	-0.3253	0.107	-3.037	0.002	-0.535	-0.115

3. Now, we want to find the probabilities of each class. We fit the test observations into each fitted model and apply the sigmoid function as $P(Y=y) = \hat{y} = \frac{e^{b_0+b_1x_1+\cdots}}{1+e^{b_0+b_1x_1+\cdots}}$

```
In [14]: #probabilities
probs=[]
for i in range(4):
    probs.append(np.round(sigmoid(x_test.dot(models_list[i].params)),4))
probs
```

```
Out[14]: [0      0.0445
 1      0.2571
 2      0.4213
 3      0.0225
 4      0.6114
         ...
 331    0.0312
 332    0.0148
 333    0.0154
 334    0.0525
 335    0.0111
 Length: 336, dtype: float64,
 0      0.8522
 1      0.3302
 2      0.7555
 3      0.0315
 4      0.5863
         ...
 331    0.0371
```

We save the predictions as a list for each observation as shown below.

```
In [15]: #predictions
predict=[]
for i in range(len(test)):
    predict.append(np.argmax(np.array((probs[0][i],probs[1][i],probs[2][i],probs[3][i])))+1)
predict
```

```
Out[15]: [2,
 2,
 2,
 4,
 1,
 1,
 2,
 1,
 1,
 2,
 1,
 2,
 1,
 2,
 1,
 1,
 1,
 2,
 1,
```

Finally, we standardize the probabilities so that the sum of a classes is equal to 1. We print the results as a txt file as shown below.

```
In [16]: #standardize probabilities
sums = sum(probs)
for i in range(4):
    probs[i]=np.round(probs[i]/sums,2)
probs
```

```
Out[16]: [0      0.05
 1      0.44
 2      0.36
 3      0.02
 4      0.51
        ...
 331    0.03
 332    0.02
 333    0.01
 334    0.05
 335    0.01
 Length: 336, dtype: float64,
 0      0.95
 1      0.56
 2      0.64
 3      0.03
 4      0.49
        ...
```

HW5_output1.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

ID, Actual class, Class 1, Class 2, Class 3, Class 4, Final prediction

---

1, 1, 0.05, 0.95, 0.0, 0.0, 2

2, 1, 0.44, 0.56, 0.0, 0.0, 2

3, 1, 0.36, 0.64, 0.0, 0.0, 2

(skip: 처음 3 줄과 마지막 3 줄만 출력시킴)

334, 4, 0.01, 0.01, 0.01, 0.96, 4

335, 4, 0.05, 0.01, 0.0, 0.94, 4

336, 4, 0.01, 0.01, 0.01, 0.97, 4

4. The confusion matrix and accuracy for the test data are calculated using the code below which is analogous to previous assignments. The accuracy is 0.762 which is obtained by the sum of the diagonal components of the confusion matrix. The corresponding output file is also shown below.

Confusion Matrix and Accuracy

In [18]:
```
#Confusion matrix
conf_arr_test=np.array([[0,0,0,0],
                [0,0,0,0],
                [0,0,0,0],
                [0,0,0,0]])
for i in range(len(test)):
    if y_test[i]==1:
        if predict[i]==1:
            conf_arr_test[0,0]+=1
        elif predict[i]==2:
            conf_arr_test[0,1]+=1
        elif predict[i]==3:
            conf_arr_test[0,2]+=1
        elif predict[i]==4:
            conf_arr_test[0,3]+=1
    elif y_test[i]==2:
        if predict[i]==1:
            conf_arr_test[1,0]+=1
        elif predict[i]==2:
            conf_arr_test[1,1]+=1
        elif predict[i]==3:
            conf_arr_test[1,2]+=1
        elif predict[i]==4:
            conf_arr_test[1,3]+=1
    elif y_test[i]==3:
        if predict[i]==1:
            conf_arr_test[2,0]+=1
        elif predict[i]==2:
            conf_arr_test[2,1]+=1
        elif predict[i]==3:
            conf_arr_test[2,2]+=1
        elif predict[i]==4:
            conf_arr_test[2,3]+=1
    elif y_test[i]==4:
        if predict[i]==1:
            conf_arr_test[3,0]+=1
        elif predict[i]==2:
            conf_arr_test[3,1]+=1
        elif predict[i]==3:
            conf_arr_test[3,2]+=1
        elif predict[i]==4:
            conf_arr_test[3,3]+=1
print(conf_arr_test)
```

```
[[49 34  1  2]
 [22 47 11  5]
 [ 0  0 85  1]
 [ 2  0  2 75]]
```

In [19]:
```
#Accuracy
acc_test = np.round(np.trace(conf_arr_test)/len(test),3)
acc_test
```

Out[19]: 0.762

HW5_output2.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

Confusion Matrix (Test)

		Predicted Class			
		1	2	3	4
Actual	0	49	34	1	2
Class	1	22	47	11	5
	2	0	0	85	1
	3	2	0	2	75

Model Summary (Test)

Overall accuracy = 0.762