

Assignment 1

자유전공학부

2019-11563

김지원

Problem 1

(a)

```

1 X_t = sm.add_constant(X)
2 reg = sm.OLS(Y, X_t).fit() # Linear Regression
3 print(reg.summary())

```

Executed at 2023.10.04 15:21:32 in 103ms

OLS Regression Results

```

=====
Dep. Variable:          Sales    R-squared:                0.239
Model:                  OLS      Adj. R-squared:            0.234
Method:                 Least Squares    F-statistic:            41.52
Date:                   Wed, 04 Oct 2023    Prob (F-statistic):      2.39e-23
Time:                   15:21:32    Log-Likelihood:         -927.66
No. Observations:       400    AIC:                    1863.
Df Residuals:           396    BIC:                    1879.
Df Model:                3
Covariance Type:        nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	13.0435	0.651	20.036	0.000	11.764	14.323
Price	-0.0545	0.005	-10.389	0.000	-0.065	-0.044
Urban	-0.0219	0.272	-0.081	0.936	-0.556	0.512
US	1.2006	0.259	4.635	0.000	0.691	1.710

```

=====

```

먼저 pandas package 를 이용하여 데이터를 불러온 뒤, qualitative variable 인 'Urban', 'US' column 의 'Yes', 'No' 를 각각 1 과 0 으로 변환하여 준다 (Use get_dummies). 이후, statsmodels api 를 이용하여 linear regression 을 진행한 뒤 summary 를 통해 R-squared 값을 구하면, 0.239 가 나온다.

(b)

	coef	std err	t	P> t	[0.025	0.975]
const	13.0435	0.651	20.036	0.000	11.764	14.323
Price	-0.0545	0.005	-10.389	0.000	-0.065	-0.044
Urban	-0.0219	0.272	-0.081	0.936	-0.556	0.512
US	1.2006	0.259	4.635	0.000	0.691	1.710

먼저, 각 coefficient 는 해당 feature 의 변화에 따른 sales 의 변화량을 나타낸다. Price 의 경우 1 증가할 때마다 sales 가 0.0545 감소한다고 볼 수 있고, Urban, US 의 경우 0 또는 1 dummy variable 로 변환하였기 때문에, 각각의 특성을 가지는 경우 (1 일 때) sales 가 0.0219 감소, 1.2006 증가한다고 볼 수 있다. 다만, const coefficient 는 predictor 와는 상관없이 intercept 의 의미를 가진다.

다음으로, p-value 를 보면, 'Urban'은 0.936 으로 매우 높고, 나머지는 0 에 가까운 낮은 값이다. 그러므로 'Price', 'US'는 통계적으로 'Sales' 예측에 유의미한 기여를 한다고 볼 수 있고, 'Urban' 특성은 'Sales' 값 예측에 통계적으로 유의미한 차이를 가져다 주지 못한다고 볼 수 있다.

(c)

Sales = 13.0435 - 0.0545 * Price - 0.0219 * Urban + 1.2006 * US 로 표현할 수 있다.

(다만, qualitative variable 인 'Urban', 'US' 값은 위 모델에서 dummy variable 로 변환하였기 때문에 Urban 과 US 값이 Yes 인 경우 1, No 인 경우 0 값을 가진다.)

(d)

'Price', 'US' 는 p-value 가 0 에 가까운 작은 값이므로 null hypothesis 를 기각할 수 있다.

'Urban'의 경우 p-value 가 1 에 가까운 매우 큰 값이므로 null hypothesis 를 기각할 수 없다.

	coef	std err	t	P> t	[0.025	0.975]
const	13.0435	0.651	20.036	0.000	11.764	14.323
Price	-0.0545	0.005	-10.389	0.000	-0.065	-0.044
Urban	-0.0219	0.272	-0.081	0.936	-0.556	0.512
US	1.2006	0.259	4.635	0.000	0.691	1.710

특히, 위 표에서 95%의 신뢰구간을 나타내는 마지막 두 column 을 본다면, ‘Urban’의 경우 coefficient 에 대한 신뢰구간이 [-0.556, 0.512] 이다. 이 신뢰구간에 0 이 포함되어 있기 때문에 null hypothesis 를 기각할 수 없다.

(e)

```
1 X = data[['Price', 'US']]
2 Y = data['Sales']
3
4 X_t = sm.add_constant(X)
5 reg = sm.OLS(Y, X_t).fit() # Linear Regression
6 print(reg.summary())
```

Executed at 2023.10.04 17:45:11 in 42ms

OLS Regression Results						
=====						
Dep. Variable:	Sales	R-squared:	0.239			
Model:	OLS	Adj. R-squared:	0.235			
Method:	Least Squares	F-statistic:	62.43			
Date:	Wed, 04 Oct 2023	Prob (F-statistic):	2.66e-24			
Time:	17:45:11	Log-Likelihood:	-927.66			
No. Observations:	400	AIC:	1861.			
Df Residuals:	397	BIC:	1873.			
Df Model:	2					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]

const	13.0308	0.631	20.652	0.000	11.790	14.271
Price	-0.0545	0.005	-10.416	0.000	-0.065	-0.044
US	1.1996	0.258	4.641	0.000	0.692	1.708
=====						

위 문제에서 sales 와 유의미한 연관성이 없다고 판별된 'Urban' 데이터를 제외하고, 'Price', 'US'만을 사용한 smaller model 을 fit 하였다. 이전 linear regression 과 R-squared 값은 0.239 로 동일하고, 각각의 coefficient 값은 조금씩 변화한 것을 볼 수 있다.

(f)

먼저, a 와 e 에서 fitting 한 model 모두 R-squared 값이 0.239 로 동일하기 때문에 이 metric 만으로는 두 모델의 성능을 비교하기 어렵다. OLS Regression Results 에 나온 다른 metric 을 통해 비교를 진행하기 위해 두 모델의 summary 를 다시 보면 다음과 같다.

(a)	R-squared: 0.239 Adj. R-squared: 0.234 F-statistic: 41.52 Prob (F-statistic): 2.39e-23 Log-Likelihood: -927.66 AIC: 1863. BIC: 1879.	(b)	R-squared: 0.239 Adj. R-squared: 0.235 F-statistic: 62.43 Prob (F-statistic): 2.66e-24 Log-Likelihood: -927.66 AIC: 1861. BIC: 1873.
-----	--	-----	--

Adjusted R-squared 값을 보면 각각 0.234, 0.235 으로 e 에서의 모델이 약간 더 큰 값을 가진다. 이는 e 에서 model size 가 더 작다는 점이 반영된 것이다.

AIC, BIC 의 경우 더 작은 값 (값이 작을수록 모델이 더 fit 을 잘하기 때문에) 역시 e 에서의 모델이기 때문에 두 번째 e 모델이 sales prediction 성능이 조금 더 높다고 할 수 있다.

다만, 두 모델 모두 R-squared 값이 0.239 이므로, Fitting 한 Model 이 Sales 값의 약 23.9% 를 설명 가능하다고 해석할 수 있다. 0.6 혹은 0.65 이상의 r-squared 값에 대해서 통상적으로 유의미한 회귀식이라고 받아들여지고 있으므로, 0.239 는 작은 값으로 a, e 두 모델 모두 전체 데이터를 잘 fit 하는 편은 아니다.

(g)

	coef	std err	t	P> t	[0.025	0.975]
const	13.0308	0.631	20.652	0.000	11.790	14.271
Price	-0.0545	0.005	-10.416	0.000	-0.065	-0.044
US	1.1996	0.258	4.641	0.000	0.692	1.708

위 summary 에서 각각의 95% 신뢰구간을 알 수 있다.

Constant : [11.790, 14.271]
Price : [-0.065, -0.044]
US : [0.692, 1.708]

혹은 Variable x 에 대한 coefficient 값을 $\hat{\beta}_x$ 이라고 했을 때, SE(standard error)를 이용하여 아래 공식을 통해 구할 수도 있다 : $[\hat{\beta}_x - 2 \times SE(\hat{\beta}_x), \hat{\beta}_x + 2 \times SE(\hat{\beta}_x)]$

Problem 2

(a)

스텝 1, 2 : 첨부한 코드에 구현되어 있다.

스텝 3 :

```
threshold = 0.5
Y_pred = (pred > threshold).astype(int)
Y_pred
```

Executed at 2023.10.04 22:06:33 in 8ms

	<unnamed>
6252	0
4684	0
1731	0
4742	0
4521	0
6340	0
576	0
5202	0
4343	0

```
# Step 4 : Compute the validation set error
1 - accuracy_score(Y_test, Y_pred)
```

Executed at 2023.10.04 22:16:18 in 12ms

0.03049999999999997

Posterior probability threshold 를 0.5 로 잡고 Y_pred 를 구한 뒤, Y_test (validation Y)와 비교하여 error 를 구하면 0.0305 정도가 나온다. Error rate 만 보면, 모델이 약 3.05%의 데이터에 대해서만 잘못 prediction 한 것으로 성능이 비교적 좋은 것으로 보인다. 그러나 실제 confusion matrix 를 통해 false negative rate 를 구해보면 약 73.91%로 성능이 매우 나쁨을 알 수 있다.

```
confusion_matrix(Y_test, Y_pred)
```

Executed at 2023.10.04 22:17:03 in 12ms

	0	1
0	1921	10
1	51	18

(Confusion Matrix)

(b)

	coef	std err	z	P> z	[0.025	0.975]
const	-11.8297	0.506	-23.387	0.000	-12.821	-10.838
balance	0.0058	0.000	22.038	0.000	0.005	0.006
income	2.075e-05	5.64e-06	3.682	0.000	9.71e-06	3.18e-05

Intercept(const), balance, income 모두 p-value 가 0 에 가까운 값이므로 통계적으로 prediction 에 유의미한 기여를 한다고 볼 수 있다.

Balance 의 coefficient 값이 0.0058 이므로 fix 된 income 값에서 balance 값이 1 unit 증가할 때 마다 Y(default)의 log odds (log(Default 일 확률 – Default 가 아닐 확률))이 0.0058 만큼 증가하게 된다. Income 의 경우 fix 된 balance 값에서 income 값이 1 unit 증가할 때마다 Y 의 log odd 가 2.07e-05 만큼 증가한다.

(c)

K-fold CV 를 구현하는 함수는 첨부한 코드에 ‘K_Fold_CV’ 함수로 구현하였다. 먼저 데이터를 random shuffling 하고, k fold 를 하기 위해 split 을 진행한다. 이후 각 portion 을 validation set, 나머지를 training set 으로 둔 뒤 training 과 validation 을 진행하여 각각의 스텝에서의 error rate 를 배열에 저장한다. 마지막으로 저장된 error rate 의 평균을 구하면 약 0.0263 이 나왔다.

코드 :

```

def K_Fold_CV (X, Y, k) :

    np.random.seed(42) # set seed
    shuffled_index = np.random.permutation(len(X))
    X = X.iloc[shuffled_index] # data shuffling
    Y = Y.iloc[shuffled_index]

    shuffled_X = X.reset_index(drop=True) # reset index
    shuffled_Y = Y.reset_index(drop=True)

    # ex) k = 5
    fold_size = int(len(X) / k)      # num of data in each fold
    split_point = [fold_size * i for i in range(k)]
    split_point.append(len(X))

    step_errors = []    # to store each step errors
    index = [i for i in range(len(shuffled_X))]

    # K-fold CV
    for i in range (k) :
        print("Cross Validation #{i} : -----".format(i+1))

        # split train and val data
        val_idx = index[split_point[i] : split_point[i+1]]
        X_val = shuffled_X.iloc[val_idx]
        Y_val = shuffled_Y.iloc[val_idx]

        train_idx = [idx for idx in index if idx not in val_idx]
        X_train = shuffled_X.iloc[train_idx]
        Y_train = shuffled_Y.iloc[train_idx]

        # add constant
        X_train = sm.add_constant(X_train)
        X_val = sm.add_constant(X_val)

        # train
        model = sm.Logit(Y_train, X_train).fit() # logistic regression model
        pred = model.predict(X_val)
        threshold = 0.5
        Y_pred = (pred > threshold).astype(int)

        # get errors
        error_rate = 1 - accuracy_score(Y_val, Y_pred)
        step_errors.append(error_rate)
        print("Error : ", error_rate)

    return sum(step_errors) / float(k)

```

CV 각 step 의 error rate 및 평균 error rate :

```

Cross Validation #1 : -----
Optimization terminated successfully.
    Current function value: inf
    Iterations 10
Error : 0.03049999999999997
Cross Validation #2 : -----
Optimization terminated successfully.
    Current function value: inf
    Iterations 10
Error : 0.024499999999999966
Cross Validation #3 : -----
Optimization terminated successfully.
    Current function value: inf
    Iterations 10
Error : 0.022499999999999964
Cross Validation #4 : -----
Optimization terminated successfully.
    Current function value: inf
    Iterations 10
Error : 0.029000000000000026
Cross Validation #5 : -----
Optimization terminated successfully.
    Current function value: inf
    Iterations 10
Error : 0.025000000000000022
0.026299999999999999

```

a 문제에서의 error rate 는 약 0.0304 였으므로 Cross validation 진행한 (c)에서의 error rate 이 조금 더 낮게 나온 것을 볼 수 있다.

(d)

먼저, student 와 default variable 의 경우 qualitative variable 이므로, CV 를 수행하기 전 dummy variable 로 모두 바꿔준 뒤, 위의 (c)에서 만든 K_Fold_CV 함수에 넣어 Cross validation 을 진행하였다. 결과는 약 0.0269 로 위의 (c)에서의 estimated test error 보다 살짝 더 높기 때문에 'student' variable 의 추가가 reduction in test error rate 에 기여하지 않을 것으로 예상된다.


```
X = data[['balance', 'income', 'student']]
Y = data['default']
```

```
print(K_Fold_CV(X, Y, 5))
```

Executed at 2023.10.05 15:38:55 in 1s 189ms

```
Cross Validation #1 : -----
Optimization terminated successfully.
      Current function value: inf
      Iterations 10
Error : 0.03049999999999997
Cross Validation #2 : -----
Optimization terminated successfully.
      Current function value: inf
      Iterations 10
Error : 0.025000000000000002
Cross Validation #3 : -----
Optimization terminated successfully.
      Current function value: inf
      Iterations 10
Error : 0.023000000000000002
Cross Validation #4 : -----
Optimization terminated successfully.
      Current function value: inf
      Iterations 10
Error : 0.02949999999999997
Cross Validation #5 : -----
Optimization terminated successfully.
      Current function value: inf
      Iterations 10
Error : 0.026499999999999968
0.02689999999999999
```