실습 2) Logistic Regression

July 4, 2022

- **default.csv 데이터** : 어느 신용카드 회사의 고객 10,000명에 대한 다음 4개 변수의 정보를 기록
  - default : 해당 고객이 자신의 debt에 대한 default 여부를 나타냄. Yes는 defaulted, No는 not defaulted 를 의미함.
  - student : 해당 고객이 학생인지 여부를 나타냄 Yes 는 학생임 No 는 학생이 아님.
  - balance : 직전 달에 카드사용액을 납부한 이후의 고객의 balance (현 시점까지 빌린 금액).
  - income : 해당 고객의 소득

[1]:
```
import pandas as pd
import numpy as np
creditcard = pd.read_csv( 'default.csv' )
creditcard.head()
```

[1]:

	default	student	balance	income
0	Yes	Yes	1486.998122	17854.39703
1	Yes	Yes	2205.799521	14271.49225
2	Yes	Yes	1774.694223	20359.50609
3	Yes	No	1889.599190	48956.17159
4	Yes	Yes	1899.390626	20655.20000

[2]:
```
creditcard['default']= creditcard['default'].map( {'Yes' : 1, 'No' : 0 } )
creditcard['student']= creditcard['student'].map( {'Yes' : 1, 'No' : 0 } )
creditcard.head()
```

[2]:

	default	student	balance	income
0	1	1	1486.998122	17854.39703
1	1	1	2205.799521	14271.49225
2	1	1	1774.694223	20359.50609
3	1	0	1889.599190	48956.17159
4	1	1	1899.390626	20655.20000

[3]:
```
creditcard.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 809 entries, 0 to 808
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   default  809 non-null    int64
```

```
 1   student   809 non-null    int64
 2   balance   809 non-null    float64
 3   income    809 non-null    float64
dtypes: float64(2), int64(2)
memory usage: 25.4 KB
```

[4]:
```
creditcard['default'].value_counts()
```

[4]:
```
0    476
1    333
Name: default, dtype: int64
```

- **sklearn.preprocession 모듈 : StandardScaler**
  - 개별 특성 변수를 평균이 0이고 표준편차가 1인 값으로 변환.
  - StandardScaler 객체를 생성한 뒤, fit(X) 메서드와 transform(X) 메서드에 변환 대상 특성 데이터 세트를 입력하여 차례로 호출.
  - 스케일 변환된 데이터 세트가 ndarray로 반환됨.

[5]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
scaler = StandardScaler()
creditcard_Xscaled = scaler.fit_transform(creditcard[[ 'student', 'balance',␣
 ↪'income' ]])
X_train, X_test, y_train, y_test=train_test_split(
    creditcard_Xscaled,
    creditcard[ 'default' ],
    test_size=0.3, random_state=0)
```

- **sklearn.linear_model 모듈: LogisticRegression (로지스틱회귀모형)**
  - 로지스틱회귀모형의 훈련 : sklearn.linear_model 모듈의 **LogisticRegression 클래스**
    - *sklearn.linear_model.LogisticRegression(penalty='l2', C=1.0, fit_intercept=True, solver='liblinear', max_iter=100)*
    - 입력 파라미터
      - solver : 최적화에 사용할 알고리즘 결정('newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga')
      - max_iter :에포크. 한번에 훈련자료 하나씩 뽑아서 가중치 업데이트, 전체 훈련 자료 몇 회 반복할지 결정.
      - penalty : 'l1','l2' 또는 'none'. 'l2'가 디폴트임. 각 특성변수 별 가중치가 커지지 못하게 함으로써 과적합을 방지.
      - C : 1.0이 디폴트. 작은 값일수록 과적합 방지
    - 메서드
      - fit(X_train,y_train) 메서드로 모델을 학습.
      - predict(X_test) 메서드로 새로운 입력데이터에 학습된 모델을 적용한 예측. 클래스 레이블을 반환.
      - predict_proba(X_test) 메서드는 각 클래스에 속할 확률을 반환.
    - 속성
      - coef_ : 가중치(기울기계수) 추정치
      - intercetp_ : 편향(절편) 추정치

[6]:
```
from sklearn.linear_model import LogisticRegression
lr_clf = LogisticRegression( penalty='none' )
lr_clf.fit(X_train, y_train)
lr_clf.coef_
```

[6]:
```
array([[-0.38319425,  4.01243062, -0.28954033]])
```

[7]:
```
coeff = pd.Series( data= np.round( lr_clf.coef_[0], decimals=4),
                   index=[ 'student','balance', 'income' ] )
coeff
```

[7]:
```
student   -0.3832
balance    4.0124
income    -0.2895
dtype: float64
```

[8]:
```
y_pred= lr_clf.predict(X_test)
y_pred[:10]
```

[8]:
```
array([0, 1, 1, 0, 1, 0, 0, 0, 1, 0], dtype=int64)
```

[9]:
```
y_pred_proba = lr_clf.predict_proba(X_test)
y_pred_proba[:10]
```

[9]:
```
array([[9.98414217e-01, 1.58578314e-03],
       [5.56087590e-03, 9.94439124e-01],
       [4.47809851e-03, 9.95521901e-01],
       [9.03839316e-01, 9.61606843e-02],
       [2.51949981e-01, 7.48050019e-01],
       [8.25459118e-01, 1.74540882e-01],
       [8.95160911e-01, 1.04839089e-01],
       [9.99858686e-01, 1.41314430e-04],
       [2.63891611e-02, 9.73610839e-01],
       [9.98550925e-01, 1.44907466e-03]])
```

- **sklearn.metric 모듈의 분류 평가를 위한 지표 API**
  - 입력파라미터
    - y_true : 실제 범주값 배열
    - y_pred : 예측 범주값 배열
    - y_pred_proba1 : 관심범주(Y=1)의 예측확률값 배열
  - accuracy_score (y_true, y_pred) : 정확도
  - confusion_matrix (y_true, y_pred) : 교차(혼동) 행렬
  - precision_score (y_true, y_pred) : 정밀도
  - recall_score (y_true, y_pred) : 재현율
  - f1_score (y_true, y_pred) : 정밀도와 재현율의 조화평균
  - precision_recall_curve (y_true, y_pred_proba_1) : 가능한 모든 임계값에 대해 정밀도와 재현율의 값을 정렬된 리스트로 반환

- roc_curve (y_true, y_pred_proba_1) : 가능한 모든 임계값에 대한 FPR과 TPR을 정렬된 리스트로 반환
- roc_auc_score (y_true, y_pred_proba_1) : ROC 커브에서의 곡선 아래 면적(AUC) 반환

[10]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
 ↪recall_score, f1_score
accuracy_score ( y_test, y_pred )
```

[10]:
```
0.8847736625514403
```

[11]:
```
confusion_matrix( y_test, y_pred )
```

[11]:
```
array([[136,  13],
       [ 15,  79]], dtype=int64)
```

[12]:
```
precision_score ( y_test, y_pred )
```

[12]:
```
0.8586956521739131
```

[13]:
```
recall_score( y_test, y_pred )
```

[13]:
```
0.8404255319148937
```

[14]:
```
f1_score( y_test, y_pred )
```

[14]:
```
0.849462365591398
```

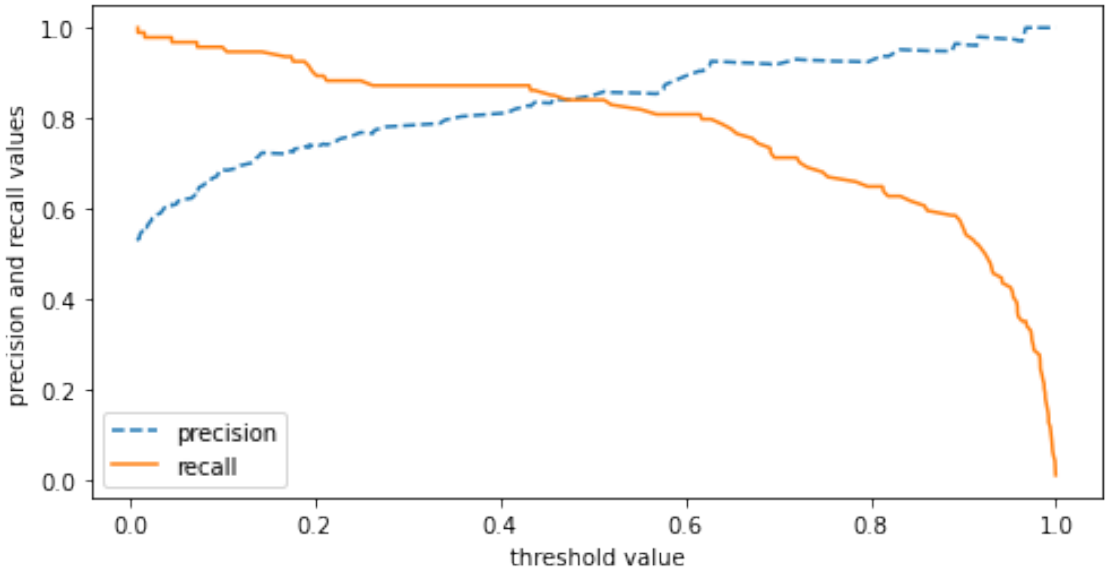
[15]:
```
from sklearn.metrics import precision_recall_curve
y_pred_proba1 = y_pred_proba[:,1]
precisions, recalls, thres = precision_recall_curve( y_test, y_pred_proba1 )
thres_n = thres.shape[0]
np.c_[ precisions[ : thres_n ], recalls[ : thres_n ], thres ] [:10]
```

[15]:
```
array([[0.53409091, 1.        , 0.00871071],
       [0.53142857, 0.9893617 , 0.00877631],
       [0.53448276, 0.9893617 , 0.00995061],
       [0.53757225, 0.9893617 , 0.0105086 ],
       [0.54069767, 0.9893617 , 0.01066104],
       [0.54385965, 0.9893617 , 0.01124646],
       [0.54705882, 0.9893617 , 0.01231105],
       [0.55029586, 0.9893617 , 0.01372781],
       [0.55357143, 0.9893617 , 0.01389965],
       [0.55688623, 0.9893617 , 0.01578174]])
```

[16]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
plt.plot( thres, precisions[ 0:thres_n ], linestyle='--', label='precision')
```

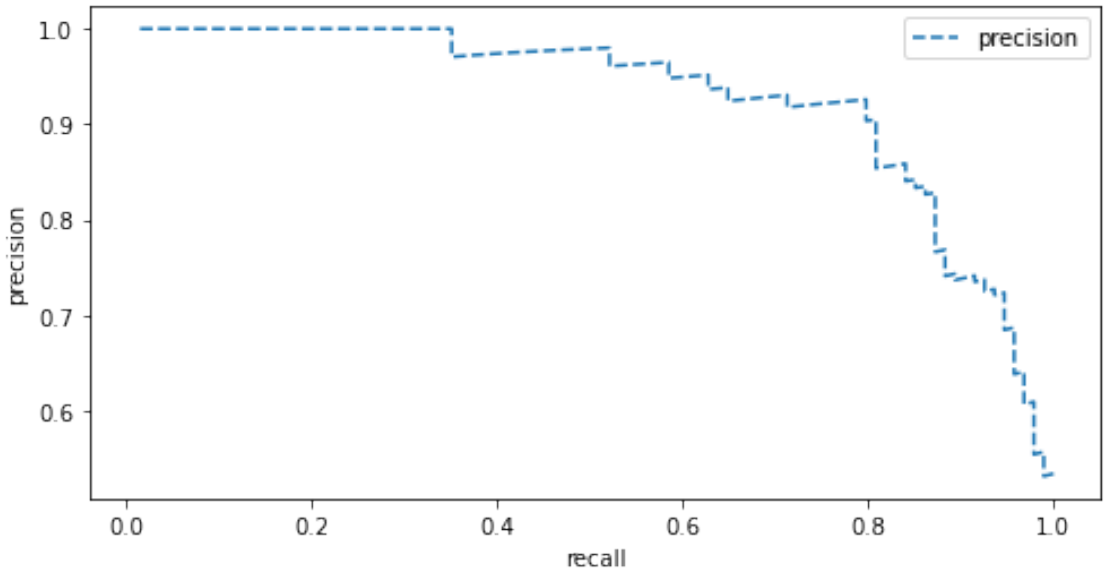
```
plt.plot( thres, recalls[ 0:thres_n ], label='recall')
plt.xlabel('threshold value')
plt.ylabel('precision and recall values')
plt.legend()
```

[16]: <matplotlib.legend.Legend at 0x374fcb4df0>



[17]:
```
plt.figure(figsize=(8,4))
plt.plot( recalls[ 0:thres_n ], precisions[ 0:thres_n ], linestyle='--',␣
 ↪label='precision')
plt.xlabel('recall')
plt.ylabel('precision')
plt.legend()
```

[17]: <matplotlib.legend.Legend at 0x374fd90e20>



[18]:
```
y_pred_new = (lr_clf.predict_proba(X_test)[:,1]>=0.6).astype(int)
y_pred_new
```

[18]:
```
array([0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0])
```

[19]:
```
confusion_matrix( y_test, y_pred_new )
```

[19]:
```
array([[140,   9],
       [ 18,  76]], dtype=int64)
```

[20]:
```
accuracy_score ( y_test, y_pred_new )
```

[20]:
```
0.8888888888888888
```

[21]:
```
precision_score( y_test, y_pred_new )
```

[21]:
```
0.8941176470588236
```

[22]:
```
recall_score( y_test, y_pred_new )
```

[22]:
```
0.8085106382978723
```

[23]:
```
f1_score(y_test, y_pred_new)
```

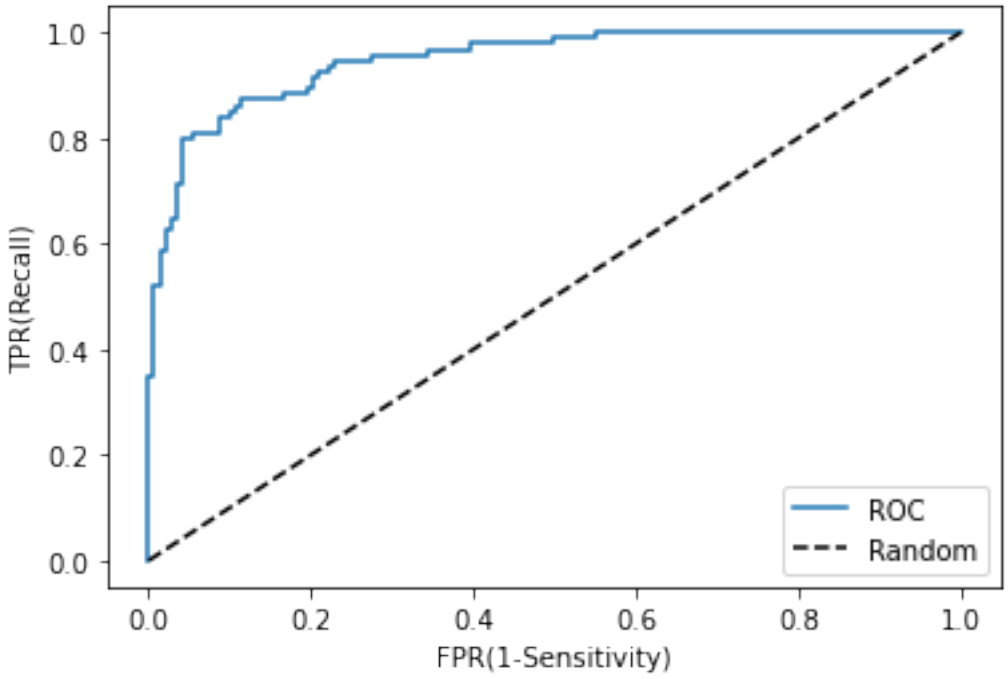
[23]:
```
0.8491620111731844
```

[24]:
```
from sklearn.metrics import roc_curve
fprs, tprs, thresholds = roc_curve( y_test, y_pred_proba1 )
np.c_[ fprs, tprs, np.round( thresholds, decimals=3 ) ][:10]
```

[24]:
```
array([[0.        , 0.        , 2.        ],
       [0.        , 0.0106383 , 1.        ],
       [0.        , 0.35106383, 0.967     ],
       [0.00671141, 0.35106383, 0.964     ],
       [0.00671141, 0.5212766 , 0.916     ],
       [0.01342282, 0.5212766 , 0.915     ],
       [0.01342282, 0.58510638, 0.891     ],
       [0.02013423, 0.58510638, 0.887     ],
       [0.02013423, 0.62765957, 0.832     ],
       [0.02684564, 0.62765957, 0.819     ]])
```

[25]:
```
plt.plot( fprs, tprs, label="ROC")
plt.plot( [0,1], [0,1],'k--', label="Random")
plt.xlabel('FPR(1-Sensitivity)')
plt.ylabel('TPR(Recall)')
plt.legend()
```

[25]:
```
<matplotlib.legend.Legend at 0x374fe42970>
```



[26]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_pred_proba1 )
```

[26]: 0.9471655004997859