머신러닝과 딥러닝

Report4

소프트웨어학과

2016312568 정희윤

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import sklearn
        from sklearn.metrics import *
        from pandas import DataFrame, Series

        plt.style.use('seaborn')
        sns.set(font_scale=2.5)
        df_train = pd.read_csv('/home/kwnam/다운로드/titanic/train.csv')
        df_test = pd.read_csv('/home/kwnam/다운로드/titanic/test.csv')
        train = df_train.drop(['Cabin', 'Embarked', 'Name', 'Ticket', 'PassengerId'],axis=1)
        test = df_test.drop(['Cabin', 'Embarked', 'Name', 'Ticket'],axis=1)
        train["Age"].fillna(train.groupby("Sex")["Age"].transform("mean"), inplace=True)
        test["Age"].fillna(test.groupby("Sex")["Age"].transform("mean"), inplace=True)
        test["Fare"].fillna(test.groupby("Sex")["Fare"].transform("median"), inplace=True)
        sex_mapping = {"male": 0, "female": 1}
        train['Sex'] = train['Sex'].map(sex_mapping)
        test['Sex'] = test['Sex'].map(sex_mapping)
        age_mean = train['Age'].mean()
        age_std = train['Age'].std()
        indexNames = train[train['Age'] < age_mean - 3*age_std].index
        train.drop(indexNames , inplace=True)
        indexNames = train[train['Age'] > age_mean + 3*age_std].index
        train.drop(indexNames , inplace=True)
        fare_mean = train['Fare'].mean()
        fare_std = train['Fare'].std()
        indexNames = train[train['Fare'] < fare_mean - 3*fare_std].index
        train.drop(indexNames , inplace=True)
        indexNames = train[train['Fare'] > fare_mean + 3*fare_std].index
        train.drop(indexNames , inplace=True)
        #'4장_Titanic dataset 탐색 및 전처리.ipynb'에서 전처리 코드만 실행하는 부분
```

< 데이터 전처리 과정 >

```
In [2]: from sklearn.linear_model import LogisticRegression
        from sklearn import metrics
        from sklearn.model_selection import train_test_split

        X_train = train.drop('Survived', axis=1).values
        target_label = train['Survived'].values
        X_test = test.values

In [3]: X_train.shape, X_test.shape

Out[3]: ((864, 6), (418, 7))

In [4]: X_tr, X_vld, y_tr, y_vld = train_test_split(X_train, target_label, test_size=0.2, random_state=2020)
        y_tr.shape, y_vld.shape

Out[4]: ((691,), (173,))

In [5]: model = LogisticRegression()
        model.fit(X_tr, y_tr)
        prediction = model.predict(X_vld)

In [6]: prediction

Out[6]: array([1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
               1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
               0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
               0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
               0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
               0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
               1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
               1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0])
```

위는 Logistic회귀를 사용하여 titanic dataset을 학습시키는 과정이다.

```
In [7]: print('Number of people: {} \naccuracy: {:.2f}% '.format(y_vld.shape[0], 100 * accuracy_score(y_vld,predictior

Number of people: 173
accuracy: 78.03%

In [8]: confusion_matrix(y_vld,prediction)

Out[8]: array([[94, 24],
               [14, 41]])

In [9]: print('Precision: {:.2f}% \nRecall: {:.2f}% \nF1-score: {:.2f}% '.format(100*precision_score(y_vld,prediction)
                                                                                100*recall_score(y_vld,prediction),1(

Precision: 63.08%
Recall: 74.55%
F1-score: 68.33%
```

위는 Cut off value를 따로 설정하지 않았을 때의 결과이다. Default value는 0.5이다.

In [10]:
```python
#cut off 조절에 따른 모델의 성능을 평가해 보기 위하여 cut off 값 생성 및 각각의 성능 지표 도출
#cut off 값은 다양하게 선택 가능.
list = []
for i in np.linspace(0,1,100):
    pred = model.predict_proba(X_vld)[:,1] > i
    cf_mtx = confusion_matrix(y_vld, pred)
    acc = accuracy_score(y_vld, pred)
    tpr = cf_mtx[0,0] / cf_mtx[0].sum()
    fpr = cf_mtx[1,0] / cf_mtx[1].sum()
    f1 = f1_score(y_vld, pred)
    list.append([i, acc, f1, tpr, fpr])

cut_off = DataFrame(list)
cut_off.columns = ["CUTOFF", "ACC", "F1", "TPR", "FPR"]
cut_off
```

Out[10]:

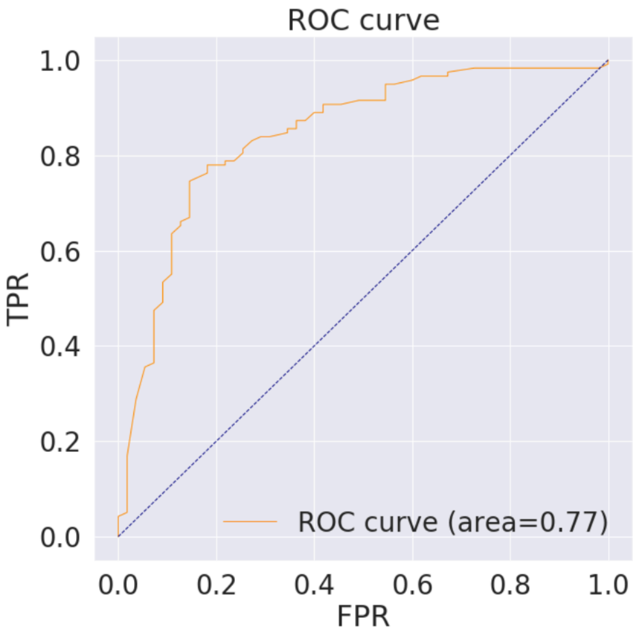
	CUTOFF	ACC	F1	TPR	FPR
0	0.000000	0.317919	0.482456	0.000000	0.000000
1	0.010101	0.323699	0.484581	0.008475	0.000000
2	0.020202	0.323699	0.484581	0.008475	0.000000
3	0.030303	0.329480	0.486726	0.016949	0.000000
4	0.040404	0.329480	0.486726	0.016949	0.000000
...	...	...	...	...	...
95	0.959596	0.687861	0.100000	0.983051	0.945455
96	0.969697	0.687861	0.100000	0.983051	0.945455
97	0.979798	0.676301	0.034483	0.983051	0.981818
98	0.989899	0.676301	0.000000	0.991525	1.000000
99	1.000000	0.682081	0.000000	1.000000	1.000000

100 rows × 5 columns

In [11]:
```python
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_vld, prediction)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10,10))
plt.plot(cut_off["FPR"],cut_off["TPR"], color="darkorange", lw=1, label="ROC curve (area=%.2f)" %roc_auc)
plt.plot([0,1], [0,1], color='navy', lw=1, linestyle='--')
plt.title("ROC curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend(loc="lower right")
```

Out[11]: <matplotlib.legend.Legend at 0x7f4675706610>



위는 다양한 Cut off value에 따른 결과값들이다.

```
In [12]: cut_off[cut_off["ACC"] == cut_off["ACC"].max()] #accuracy가 최대인 값
```

Out[12]:

	CUTOFF	ACC	F1	TPR	FPR
70	0.707071	0.803468	0.653061	0.90678	0.418182

```
In [13]: cut_off_ACC_MAX = cut_off[cut_off["ACC"] == cut_off["ACC"].max()]["CUTOFF"][70]
         cut_off_ACC_MAX
Out[13]: 0.7070707070707072

In [14]: pred_ACC_MAX = model.predict_proba(X_vld)[:,1] > cut_off_ACC_MAX

In [15]: confusion_matrix(y_vld,pred_ACC_MAX)
Out[15]: array([[107,  11],
               [ 23,  32]])

In [16]: cut_off[cut_off["F1"] == cut_off["F1"].max()] #F1-score가 최대인 값
```

Out[16]:

	CUTOFF	ACC	F1	TPR	FPR
45	0.454545	0.791908	0.714286	0.779661	0.181818

```
In [17]: cut_off_F1_MAX = cut_off[cut_off["F1"] == cut_off["F1"].max()]["CUTOFF"][45]
         cut_off_F1_MAX
Out[17]: 0.4545454545454546

In [18]: pred_F1_MAX = model.predict_proba(X_vld)[:,1] > cut_off_F1_MAX

In [19]: confusion_matrix(y_vld,pred_F1_MAX)
Out[19]: array([[92, 26],
               [10, 45]])
```

위는 최적의 Cut off value를 찾는 방법이다.

위에서 볼 수 있다시피 Accuracy가 최대일 때와 F1-score가 최대일 때 Cut off value가 중요하다. 둘중에 어떤것을 사용할지는 목적에 따라 달라진다.