캐글 코리아 4차 대회

학습 내용

- 라벨 인코딩 적용
- 다양한 모델 성능 비교
- 이진 분류 평가지표 활용

목차

01. 라이브러리 임포트 및 데이터 준비

[목차로 이동하기](#)

In [1]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import warnings

warnings.filterwarnings('ignore')
```

In [2]:

```
train = pd.read_csv('data/4th_kaggle/train.csv')
test = pd.read_csv('data/4th_kaggle/test.csv')
sub = pd.read_csv('data/4th_kaggle/sample_submission.csv')
```

데이터 탐색

- 컬럼명 : [].columns
- 행열 : [].shape
- 정보 : [].info()
- 수치 데이터 요약정보 : [].describe()
- 결측치 : [].isnull().sum()

데이터 정보

age : 나이
workclass : 고용 형태
fnlwgt : 사람 대표성을 나타내는 가중치 (final weight의 약자)
education : 교육 수준 (최종 학력)
education_num : 교육 수준 수치
marital_status: 결혼 상태
occupation : 업종
relationship : 가족 관계
race : 인종
sex : 성별
capital_gain : 양도 소득
capital_loss : 양도 손실
hours_per_week : 주당 근무 시간
native_country : 국적
income : 수익 (예측해야 하는 값, target variable)

In [4]:

```
print("학습용 데이터 : ", train.shape)
print("테스트용 데이터 : ", test.shape)
```

```
학습용 데이터 :  (26049, 16)
테스트용 데이터 :  (6512, 15)
```

In [6]:

```
y = train['income']
test['income'] = "blank"

all_dat = pd.concat([train, test], axis=0)
print(all_dat.shape)
```

```
(32561, 16)
```

In [7]:

```
all_dat.income.value_counts()
```

Out[7]:

```
<=50K     19744
blank      6512
>50K       6305
Name: income, dtype: int64
```

02. 데이터 전처리

목차로 이동하기

In [8]:

```
all_dat.loc[ all_dat['income']=='>50K' , 'target'] = 1
all_dat.loc[ all_dat['income']=='<=50K' , 'target'] = 0
all_dat.loc[ all_dat['income']=='blank' , 'target'] = 999
all_dat['target'] = all_dat.target.astype("int")
```

라벨 인코딩

In [9]:

```
from sklearn.preprocessing import LabelEncoder
```

In [10]:

```
en_x = LabelEncoder()
all_dat['workclass_lbl'] = en_x.fit_transform(all_dat['workclass'])
all_dat.head(3)
```

Out[10]:

	id	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relations
0	0	40	Private	168538	HS-grad	9	Married-civ-spouse	Sales	Husb
1	1	17	Private	101626	9th	5	Never-married	Machine-op-inspct	Own-c
2	2	18	Private	353358	Some-college	10	Never-married	Other-service	Own-c

In [11]:

```
all_dat['education_lbl'] = en_x.fit_transform(all_dat['education'])
all_dat['marital_status_lbl'] = en_x.fit_transform(all_dat['marital_status'])
all_dat['occupation_lbl'] = en_x.fit_transform(all_dat['occupation'])
all_dat['relationship_lbl'] = en_x.fit_transform(all_dat['relationship'])
all_dat['race_lbl'] = en_x.fit_transform(all_dat['race'])
all_dat['native_country_lbl'] = en_x.fit_transform(all_dat['native_country'])
all_dat.head(3)
```

Out[11]:

	id	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relations
0	0	40	Private	168538	HS-grad	9	Married-civ-spouse	Sales	Husba
1	1	17	Private	101626	9th	5	Never-married	Machine-op-inspct	Own-cl
2	2	18	Private	353358	Some-college	10	Never-married	Other-service	Own-cl

3 rows × 24 columns

In [12]:

```
mf_mapping = {"Male": 1, "Female": 2}
all_dat['sex'] = all_dat['sex'].map(mf_mapping)
all_dat.head(3)
```

Out[12]:

	id	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relations
0	0	40	Private	168538	HS-grad	9	Married-civ-spouse	Sales	Husba
1	1	17	Private	101626	9th	5	Never-married	Machine-op-inspct	Own-cl
2	2	18	Private	353358	Some-college	10	Never-married	Other-service	Own-cl

3 rows × 24 columns

In [13]:

```
sel_cat = ['workclass', 'education', 'marital_status',
           'occupation', 'relationship', 'race', 'native_country', 'income' ]
all_dat_n = all_dat.drop(sel_cat, axis=1)
all_dat_n
```

Out[13]:

	id	age	fnlwgt	education_num	sex	capital_gain	capital_loss	hours_per_week	targ
0	0	40	168538	9	1	0	0	60	
1	1	17	101626	5	1	0	0	20	
2	2	18	353358	10	1	0	0	16	
3	3	21	151158	10	2	0	0	25	
4	4	24	122234	10	2	0	0	20	
...	...	...	...	...	...	...	...	...	
6507	6507	35	61343	13	1	0	0	40	99
6508	6508	41	32185	13	1	0	0	40	99
6509	6509	39	409189	3	1	0	0	40	99
6510	6510	35	180342	9	1	0	0	40	99
6511	6511	28	156819	9	2	0	0	36	99

32561 rows × 16 columns

In [14]:

```
X_cat = all_dat_n.drop(['target'],axis=1)
y = all_dat_n['target']
```

In [16]:

```
train_n = all_dat_n.loc[ (all_dat_n['target']==0) | (all_dat_n['target']==1)  , : ]
test_n = all_dat_n.loc[ all_dat_n['target']==999  ,  : ]

print(train_n.shape, test_n.shape)
```

```
(26049, 16) (6512, 16)
```

In [17]:

```
from sklearn.model_selection import train_test_split

sel = ['age', 'education_num', 'sex']

X_tr_all = train_n[sel]
y_tr_all = train_n['target']
X_test_all = test_n[sel]

X_train, X_test, y_train, y_test = train_test_split(X_tr_all,
                                                    y_tr_all,
                                                    test_size=0.3,
                                                    random_state=77)
```

03. 모델 구축 및 평가하기

목차로 이동하기

로지스틱 모델

In [22]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [21]:

```
model = LogisticRegression()
model.fit(X_train, y_train)
pred_log = model.predict(X_test)

model.score(X_train, y_train), model.score(X_test, y_test),
```

Out[21]:

```
(0.7960403641548756, 0.7901471529110684)
```

혼동 행렬

In [23]:

```
confusion = confusion_matrix(y_test, pred_log)
print("오차 행렬:₩n{}".format(confusion))
```

```
오차 행렬:
[[5509  380]
 [1260  666]]
```

In [24]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_log,
                           target_names=["50K>=", "50K<"]))
```

```
              precision    recall  f1-score   support

       50K>=       0.81      0.94      0.87      5889
        50K<       0.64      0.35      0.45      1926

    accuracy                           0.79      7815
   macro avg       0.73      0.64      0.66      7815
weighted avg       0.77      0.79      0.77      7815
```

- f1-score 0.45

In [25]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [26]:

```
model = RandomForestClassifier()
model.fit(X_train, y_train)
pred_rf = model.predict(X_test)

model = AdaBoostClassifier()
model.fit(X_train, y_train)
pred_ada = model.predict(X_test)

model = GradientBoostingClassifier()
model.fit(X_train, y_train)
pred_gr = model.predict(X_test)
```

In [27]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_rf,
                           target_names=["50K>=", "50K<"]))
```

```
              precision    recall  f1-score   support

       50K>=       0.82      0.91      0.86      5889
        50K<       0.59      0.40      0.48      1926

    accuracy                           0.78      7815
   macro avg       0.71      0.65      0.67      7815
weighted avg       0.77      0.78      0.77      7815
```

In [28]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_ada,
                           target_names=["50K>=", "50K<"]))
```

```
              precision    recall  f1-score   support

       50K>=       0.82      0.94      0.88      5889
        50K<       0.67      0.36      0.47      1926

    accuracy                           0.80      7815
   macro avg       0.74      0.65      0.67      7815
weighted avg       0.78      0.80      0.78      7815
```

In [29]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_gr,
                           target_names=["50K>=", "50K<"]))
```

```
              precision    recall  f1-score   support

       50K>=       0.83      0.93      0.87      5889
        50K<       0.64      0.40      0.50      1926

    accuracy                           0.80      7815
   macro avg       0.74      0.67      0.68      7815
weighted avg       0.78      0.80      0.78      7815
```

- GradientBoostingClassifier() 알고리즘이 가장 좋은 f1-score의 값을 갖는다. 0.50