앙상블 실습

Majority Voting / Bagging / Boosting

데이터 설명

- 데이터셋: 유니버셜 은행
- 개인대출 제안에 대한 수락 여부
- 총 데이터: 5000개 (학습: 3000개, 테스트 2000개)
- 성공율: 9.6% (480명)

나이, 경력, 소득, 가족 수, 신용카드 월평균 사용 액, 교육, 담보 부채권, **개인대출**, 증권계좌, CD계좌, 온라인 뱅킹, 신용카드

Age	Experience	Income	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
25	1	49	4	1.6	1	0	0	1	0	0	0
45	19	34	3	1.5	1	0	0	1	0	0	0
39	15	11	1	1.0	1	0	0	0	0	0	0
35	9	100	1	2.7	2	0	0	0	0	0	0
35	8	45	4	1.0	2	0	0	0	0	0	1

위의 특성 변수를 이용하여 개인대출 가능 여부를 예측하는 분류기를 설계하는 문제

투표 방식(Voting) 실습 #1 (분류)

- ## 데이터 로더

```python
import pandas as pd

bank_df = pd.read_csv('UniversalBank.csv')
bank_df.head()
```

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

- ## 특성 변수 선택

```python
X = bank_df.drop (['ID','ZIPCode','PersonalLoan'], axis=1)
y = bank_df['PersonalLoan']
```

- ## 범주형 데이터 변환

```python
from sklearn.preprocessing import LabelEncoder
classle = LabelEncoder()
y = classle.fit_transform(y)

```

- ## 데이터 분할

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

투표 방식(Voting) 실습 #1 (분류)

- 앙상블로 사용할 개별 모델 정의

```
from sklearn.tree import DecisionTreeClassifier # 결정 트리
from sklearn.neighbors import KNeighborsClassifier # K-최근접 이웃
from sklearn.linear_model import LogisticRegression # 로지스틱 회귀 모델


logistic = LogisticRegression(solver='liblinear',
                              penalty='l2',
                              C=0.001,
                              random_state=1)

tree = DecisionTreeClassifier(max_depth=None,
                              criterion='entropy',
                              random_state=1)

knn = KNeighborsClassifier(n_neighbors=1,
                           p=2,
                           metric='minkowski')

```

- 앙상블-Voting 정의

```
from sklearn.ensemble import VotingClassifier # 과반수 투표(Majority Voting)
voting_estimators = [('logistic', logistic), ('tree', tree), ('knn', knn)]
voting = VotingClassifier(estimators = voting_estimators,
                          voting='soft')
```

투표 방식(Voting) 실습 #1 (분류)

- K-fold 교차 검증

```
[ ]
from sklearn.model_selection import cross_val_score # 교차타당도 # 추가

clf_labels = ['Logistic regression', 'Decision tree', 'KNN', 'Majority voting']
all_clf = [logistic, tree, knn, voting]

for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf,X=X_train,y=y_train,cv=10,scoring='roc_auc')
    print("ROC AUC: %0.3f (+/- %0.3f) [%s]", (scores.mean(), scores.std(), label))
```

```
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9276195033668649, 0.019846304471857O5, 'Logistic regression')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9499227861055811, 0.032735680131811405, 'Decision tree')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.7120816883018249, 0.04722587272864442, 'KNN')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9724206139059355, 0.01593603549077189, 'Majority voting')
```

투표 방식(Voting) 실습 #1 (분류)

- GridSearch방식을 이용한 모델 최적화

```
from sklearn.model_selection import GridSearchCV # 하이퍼파라미터 튜닝

params = {'logistic__C': [0.001, 0.1, 100.0],
          'tree__max_depth': [1, 3, 5],
          'knn__n_neighbors': [1, 3, 5]}

grid = GridSearchCV(estimator=voting,
                    param_grid=params,
                    cv=10,
                    scoring='roc_auc',
                    iid=False)
grid.fit(X_train, y_train)

for r, _ in enumerate(grid.cv_results_['mean_test_score']):
    print("%0.3f +/- %0.3f %r"
          % (grid.cv_results_['mean_test_score'][r],
             grid.cv_results_['std_test_score'][r] / 2.0,
             grid.cv_results_['params'][r]))

print('최적의 파타미터: %s' % grid.best_params_)
print('ACU: %.3f' % grid.best_score_)
```

Scoring	Function
Classification	
'accuracy'	metrics.accuracy_score
'balanced_accuracy'	metrics.balanced_accuracy_score
'average_precision'	metrics.average_precision_score
'neg_brier_score'	metrics.brier_score_loss
'f1'	metrics.f1_score
'f1_micro'	metrics.f1_score
'f1_macro'	metrics.f1_score
'f1_weighted'	metrics.f1_score
'f1_samples'	metrics.f1_score
'neg_log_loss'	metrics.log_loss
'precision' etc.	metrics.precision_score
'recall' etc.	metrics.recall_score
'jaccard' etc.	metrics.jaccard_score
'roc_auc'	metrics.roc_auc_score
'roc_auc_ovr'	metrics.roc_auc_score
'roc_auc_ovo'	metrics.roc_auc_score
'roc_auc_ovr_weighted'	metrics.roc_auc_score
'roc_auc_ovo_weighted'	metrics.roc_auc_score
Clustering	
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score
'adjusted_rand_score'	metrics.adjusted_rand_score
'completeness_score'	metrics.completeness_score
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score
'homogeneity_score'	metrics.homogeneity_score
'mutual_info_score'	metrics.mutual_info_score
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score
'v_measure_score'	metrics.v_measure_score
Regression	
'explained_variance'	metrics.explained_variance_score
'max_error'	metrics.max_error
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_root_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance

```
최적의 파타미터: {'knn__n_neighbors': 3, 'logistic__C': 100.0, 'tree__max_depth': 5}
ACU: 0.986
```

배깅 방식(Bagging) 실습 #2 (분류)

- ## 데이터 로더

```python
import pandas as pd

bank_df = pd.read_csv('UniversalBank.csv')
bank_df.head()
```

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

- ## 특성 변수 선택

```python
X = bank_df.drop (['ID','ZIPCode','PersonalLoan'], axis=1)
y = bank_df['PersonalLoan']
```

- ## 범주형 데이터 변환

```python
from sklearn.preprocessing import LabelEncoder
classle = LabelEncoder()
y = classle.fit_transform(y)

```

- ## 데이터 분할

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

배깅 방식(Bagging) 실습 #2 (분류)

- **앙상블로 사용할 개별 모델 정의**

```python
from sklearn.tree import DecisionTreeClassifier # 결정 트리

tree = DecisionTreeClassifier(max_depth=None,
                              criterion='entropy',
                              random_state=1)

```

- **앙상블-Bagging 정의**

```python
from sklearn.ensemble import BaggingClassifier # 배깅(Bagging)

bagging = BaggingClassifier(base_estimator=tree, # 수정
                            n_estimators=500,
                            max_samples=1.0,
                            max_features=1.0,
                            bootstrap=True,
                            bootstrap_features=False,
                            n_jobs=1,
                            random_state=1)
```

배깅 방식(Bagging) 실습 #2 (분류)

- K-fold 교차 검증

```
[ ]
from sklearn.model_selection import cross_val_score # 교차타당도 # 추가

clf_labels = ['Decision tree', 'Bagging']
all_clf = [tree, bagging]

for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf,X=X_train,y=y_train,cv=10,scoring='roc_auc')
    print("ROC AUC: %0.3f (+/- %0.3f) [%s]", (scores.mean(), scores.std(), label))
```

```
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9499227861055811, 0.032735680131811405, 'Decision tree')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9976668161065998, 0.001775473982951, 'Bagging')
```

부스팅 방식(Boosting) 실습 #3 (분류)

- **데이터 로더**

```python
import pandas as pd

bank_df = pd.read_csv('UniversalBank.csv')
bank_df.head()
```

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

- **특성 변수 선택**

```python
X = bank_df.drop (['ID','ZIPCode','PersonalLoan'], axis=1)
y = bank_df['PersonalLoan']
```

- **범주형 데이터 변환**

```python
from sklearn.preprocessing import LabelEncoder
classle = LabelEncoder()
y = classle.fit_transform(y)

```

- **데이터 분할**

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

부스팅 방식(Boosting) 실습 #3 (분류)

- ## 앙상블로 사용할 개별 모델 정의

```
from sklearn.tree import DecisionTreeClassifier # 결정 트리

# 배깅과 차이점: mex_depth =1 로 변경
tree = DecisionTreeClassifier(max_depth=1,
                              criterion='entropy',
                              random_state=1)


```

- ## 앙상블-Boosting 정의

```
[ ]
from sklearn.ensemble import AdaBoostClassifier # 부스팅(Boosting)

adaboost = AdaBoostClassifier(base_estimator=tree, # 수정
                              n_estimators=500,
                              learning_rate = 0.1, # 수정
                              random_state=1)

```

부스팅 방식(Boosting) 실습 #3 (분류)

- K-fold 교차 검증

```
[ ]
from sklearn.model_selection import cross_val_score # 교차타당도 # 추가

clf_labels = ['Decision tree', 'Ada boost']
all_clf = [tree, adaboost]
for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf,X=X_train,y=y_train,cv=10,scoring='roc_auc')
    print("ROC AUC: %0.3f (+/- %0.3f) [%s]", (scores.mean(), scores.std(), label))
```

```
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.8829135713666967, 0.023406169666276122, 'Decision tree')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9835566978095359, 0.010774714837632118, 'Ada boost')
```