인공지능 실습 과제

2017150003
권순호

필요한 모듈 및 라이브러리 불러오기

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from graphviz import Source
from IPython.display import Image

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn.metrics import accuracy_score,recall_score, precision_score
from sklearn.metrics import confusion_matrix, roc_curve,f1_score, auc
from sklearn.metrics import classification_report,roc_auc_score

sns.set_style("whitegrid")

from IPython.display import display, HTML
import scipy as sp
import scipy.stats as stats

import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true-y_pred)/y_true))*100

# Decision Tree
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import confusion_matrix, roc_curve,f1_score, auc

# Bagging
from sklearn.ensemble import BaggingRegressor, BaggingClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression

# Random Forest
import os
from sklearn.ensemble import RandomForestClassifier

# XGboost
import xgboost as xgb
from sklearn.ensemble import GradientBoostingRegressor, GradientBoostingClassifier

# AdaBoost
from sklearn.ensemble import AdaBoostClassifier
```

먼저, 필요한 모듈 및 라이브러리를 불러옵니다.

데이터 불러오기

In [3]:
```
data = pd.read_csv("p2p_data.csv")
data.head()
```

Out [3]:

	annual_inc	bc_util	chargeoff_within_12_mths	dti	inq_last_6mths	mths_since_last_delinq
0	53000.0	88.8	0	25.25	0	11
1	60000.0	94.4	0	25.88	0	20
2	125000.0	82.5	0	10.93	1	77
3	90000.0	77.6	0	22.75	0	79
4	76863.0	43.3	0	19.53	1	34

Read_csv()를 통해서 데이터를 불러와줍니다.

명목형 변수 제거

In [4]:
```
data = data.drop(['verification_status','home_ownership','term'],axis=1)
data.head()
```

Out [4]:

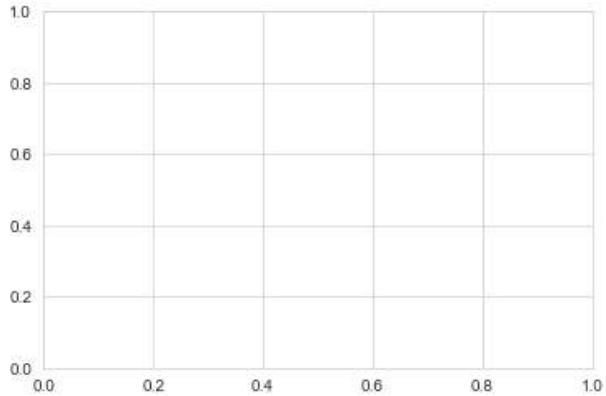
	annual_inc	bc_util	chargeoff_within_12_mths	dti	inq_last_6mths	mths_since_last_delinq
0	53000.0	88.8	0	25.25	0	11
1	60000.0	94.4	0	25.88	0	20
2	125000.0	82.5	0	10.93	1	77
3	90000.0	77.6	0	22.75	0	79
4	76863.0	43.3	0	19.53	1	34

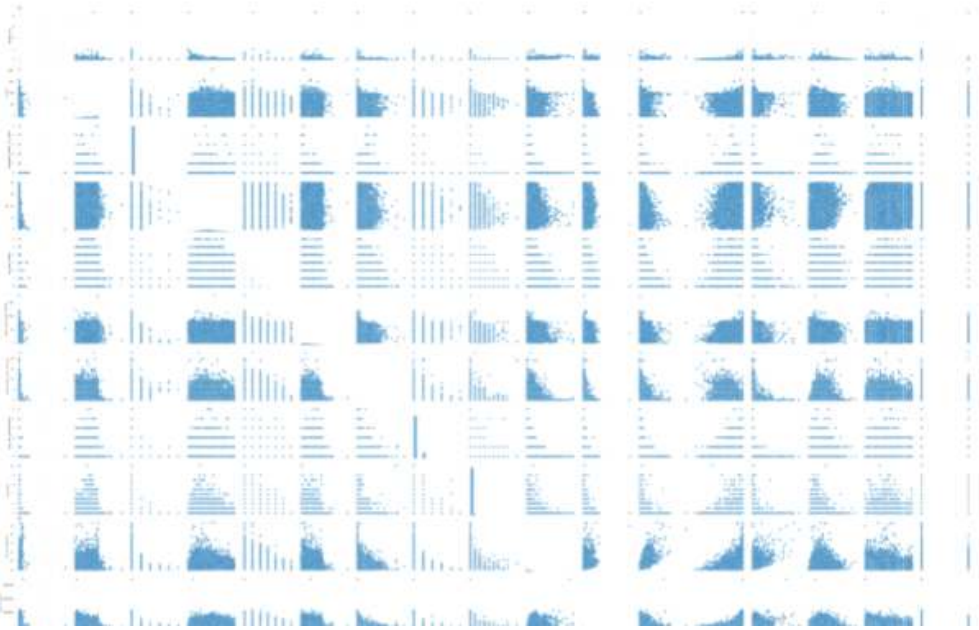
명목형 변수(의미 없는 열이나 문자열)를 drop()을 통해서 제거해줍니다.
Axis = 1은 열, axis = 0이면 행을 의미합니다.
대괄호를 통하여 한 번에 제거할 수 있습니다.

데이터를 분리하기 전에 데이터에 대해 간단히 알아보고 가겠습니다.

pairplot을 이용하여 히스토그램 및 산점도 그리기

In [55]:
```
fig, ax = plt.subplots()
sns.pairplot(data = data)
plt.savefig('pairplot.png')
```

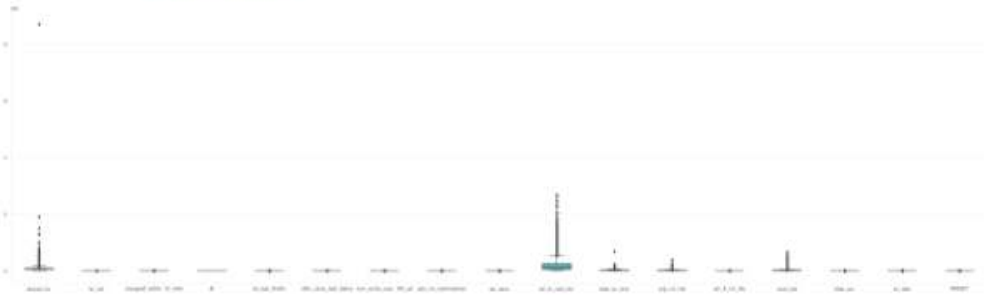




pairplot을 이용하여 히스토그램 및 산점도를 확인합니다.

boxplot을 이용하여 설명변수의 특성 파악

In [56]:
```
fig, ax = plt.subplots(figsize = (35,10))
boxplot = sns.boxplot(data = data, ax = ax, width = 0.5)
boxplot.tick_params(labelsize = 10)
plt.savefig('boxplot.png')
```

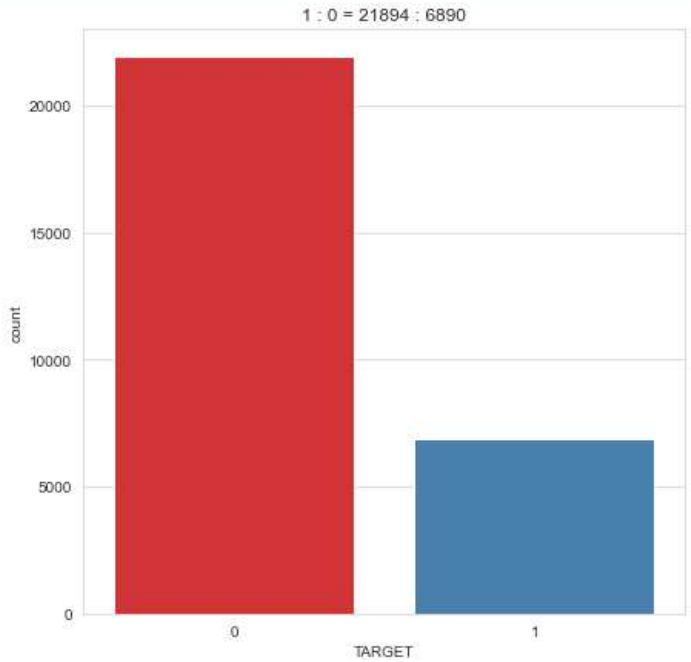


boxplot을 이용하여 설명변수의 특성을 파악합니다.

Class distribution 확인

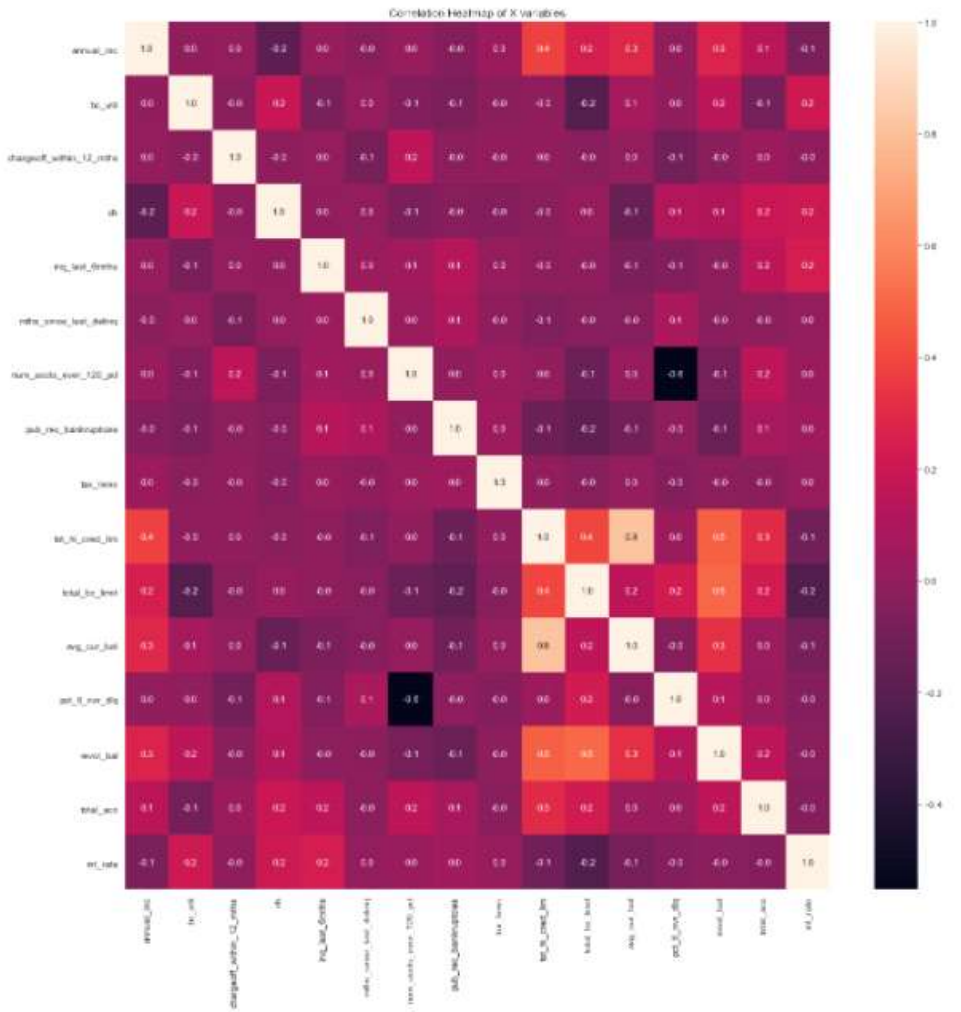
In [57]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (7,7))
sns.countplot(data['TARGET'], palette = 'Set1', ax = ax)
ax.set_title("1 : 0 = {} : {}".format(*data['TARGET'].value_counts()))
plt.show()
```

```
C:₩Users₩tnsgh₩Anaconda3₩lib₩site-packages₩seaborn₩_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```



Class distribution을 확인하기 위해 subplots을 이용해줍니다.
1 : 0의 비율은 21894 : 6890입니다.

In [58]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (15,15))
ax = sns.heatmap(data.drop(['TARGET'],axis=1).corr(), annot=True, fmt='.1f')
ax.set_title("Correlation Heatmap of X variables")
plt.tight_layout()
plt.show(fig)
```



변수간 상관관계를 알아보기 위해 heatmap을 이용해줍니다.

X와 y 분리

In [5]:
```
y = data['TARGET'].values
X = data.drop('TARGET',axis = 1)
xcolumns = X.columns.values
X = X.values
```

데이터를 8:2로 분할하여 train, test 데이터 구축

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=
```

In [39]:
```
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size = 0
```

이제, 데이터를 8:2로 분할하여 train data와 test data로 분리해줍니다.
20%를 테스트 데이터로 사용하기 위해 test_size = 0.2로 설정하여 주었고, random_state는 random seed를 무엇으로 정하느냐는 의미이므로, 어떤 실수 값을 넣어도 무관합니다.

이제 Decision Tree, Bagging, Random Forest, Adaboost, Gradient Boosting을 순서대로 사용하여 비교해보겠습니다.

Decision Tree

```
In [8]: clf_dt = DecisionTreeClassifier(criterion = 'gini', max_depth = None)
        clf_dt.fit(X_train, y_train)
```

```
Out[8]: DecisionTreeClassifier()
```

셋팅하고 fit함수를 통해 학습시켜줍니다.
Criterion은 트리를 split해줄 때의 cost-function을 의미합니다.
Max_depth를 none으로 설정하여, 모든 노드가 pure해질때까지 학습하는 fully-growned-tree로 해주었습니다.

```
In [9]: y_train_pred = clf_dt.predict(X_train)
        y_valid_pred = clf_dt.predict(X_valid)

        print('- Accuracy (Train) : {:.4}'.format(accuracy_score(y_train, y_train_pred)))
        print('- Accuracy (Validation) : {:.4}'.format(accuracy_score(y_valid, y_valid_pred)))

        print('- F1 score (Train) : {:.4}'.format(f1_score(y_train, y_train_pred)))
        print('- F1 score (Validation) : {:.4}'.format(f1_score(y_valid, y_valid_pred)))

        print('- precision score(Train) : {:.4}'.format(precision_score(y_train, y_train_pred)
        print('- precision score(Validation) : {:.4}'.format(precision_score(y_valid, y_valid_

        print('- recall score (Train) : {:.4}'.format(recall_score(y_train, y_train_pred)))
        print('- recall score (Validation) : {:.4}'.format(recall_score(y_valid, y_valid_pred)
```

```
- Accuracy (Train) : 1.0
- Accuracy (Validation) : 0.6496
- F1 score (Train) : 1.0
- F1 score (Validation) : 0.3091
- precision score(Train) : 1.0
- precision score(Validation) : 0.2928
- recall score (Train) : 1.0
- recall score (Validation) : 0.3273
```

train에 대한 값이 모두 1이 나오는 이유는 max_depth를 none으로 설정하여, fully-growned-tree가 되었기 때문에 train-data까지 모두 설명하게 되었기 때문입니다.

```
In [10]: clf_dt = DecisionTreeClassifier(criterion = 'gini', max_depth = 5)
         clf_dt.fit(X_train, y_train)
```

```
Out[10]: DecisionTreeClassifier(max_depth=5)
```

그래서 이번에는 max_depth를 5로 설정해주었습니다.

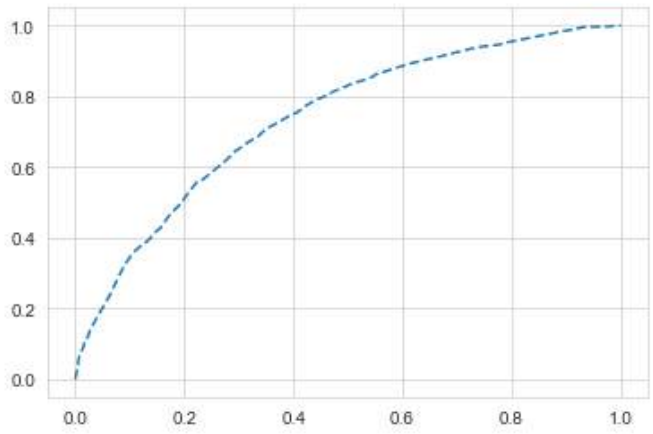
In [11]:
```
y_train_pred = clf_dt.predict(X_train)
y_valid_pred = clf_dt.predict(X_valid)

print('- Accuracy (Train) : {:.4}'.format(accuracy_score(y_train, y_train_pred)))
print('- Accuracy (Validation) : {:.4}'.format(accuracy_score(y_valid, y_valid_pred)))

print('- F1 score (Train) : {:.4}'.format(f1_score(y_train, y_train_pred)))
print('- F1 score (Validation) : {:.4}'.format(f1_score(y_valid, y_valid_pred)))

print('- precision score(Train) : {:.4}'.format(precision_score(y_train, y_train_pred)
print('- precision score(Validation) : {:.4}'.format(precision_score(y_valid, y_valid_

print('- recall score (Train) : {:.4}'.format(recall_score(y_train, y_train_pred)))
print('- recall score (Validation) : {:.4}'.format(recall_score(y_valid, y_valid_pred)
```

```
- Accuracy (Train) : 0.7682
- Accuracy (Validation) : 0.7597
- F1 score (Train) : 0.1487
- F1 score (Validation) : 0.1304
- precision score(Train) : 0.6145
- precision score(Validation) : 0.4882
- recall score (Train) : 0.0846
- recall score (Validation) : 0.07525
```

결과는 위와 같이 나옵니다.

In [100]:
```
# Show ROC Curve
fpr, tpr, thresholds = roc_curve(y_valid, y_score_valid[:,1], pos_label = 1)
plt.plot(fpr, tpr, linestyle='--')
```

Out[100]:
```
[<matplotlib.lines.Line2D at 0x19e45c7d310>]
```



ROC Curve를 그려보았습니다.

```
In [101]: print(auc(fpr,tpr))
          0.7373983807170593
```

AUC score를 계산하면 0.73 정도가 나옵니다.

```
In [123]: print('- balanced_accuracy_score (Test) : {:.4}'.format(balanced_accuracy_score(y_test
          - balanced_accuracy_score (Test) : 0.5
```

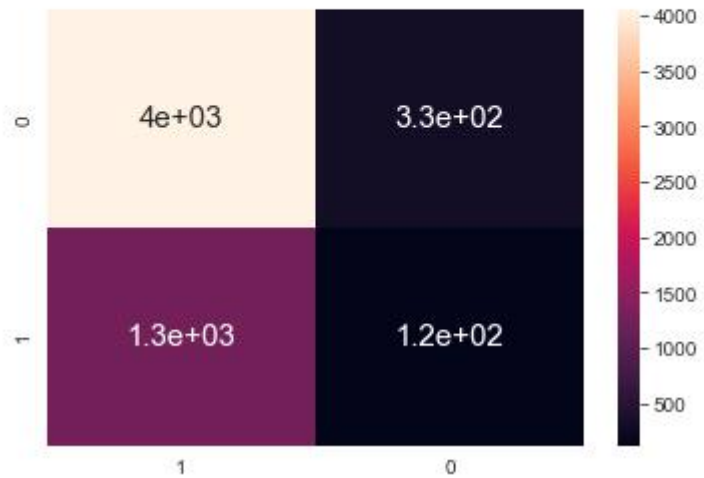
균형정확도를 계산하면 0.5가 나옵니다.

```
In [124]: print('- accuracy_score (Test) : {:.4}'.format(accuracy_score(y_test,pred_v)))
          - accuracy_score (Test) : 0.7606
```

정확도는 76.06%가 나왔습니다.

```
In [125]: cm_test = confusion_matrix(y_test, y_test_pred)
          cm_test
Out[125]: array([[4048,  331],
                 [1258,  120]], dtype=int64)
```

```
In [126]: cm_test = confusion_matrix(y_test, y_test_pred)
          cm_test = pd.DataFrame(cm_test, columns = [1,0])
          sns.heatmap(data=cm_test, annot=True, annot_kws={'size':15})
Out[126]: <AxesSubplot:>
```



모델에 대한 confusion matrix를 구하고 그림으로 나타내었습니다.

아래는 결과를 visualization하는 과정과 결과를 나타내었습니다.

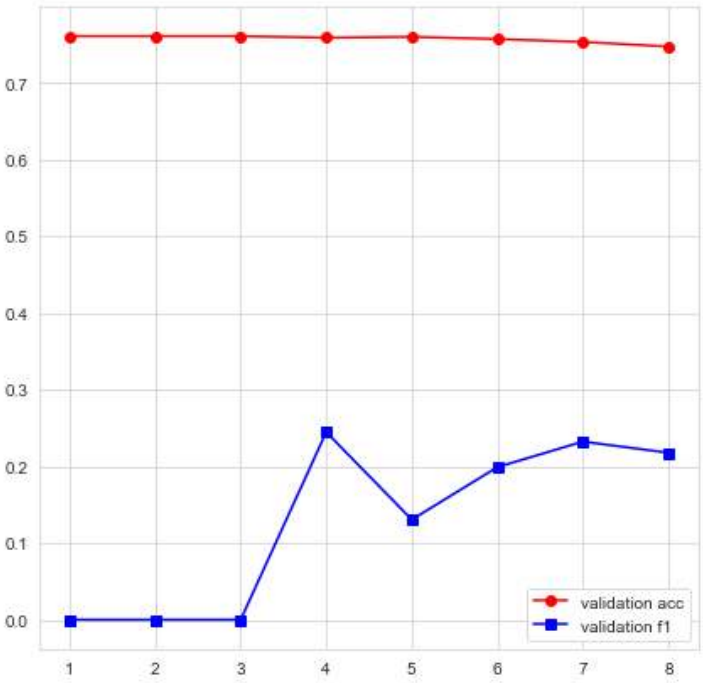
Visualization

In [15]:
```
graph = Source(export_graphviz(clf_dt,
                               out_file=None,
                               filled=True,
                               leaves_parallel=False))
Image(graph.pipe(format='png'))
```

Out[15]:

In [18]:
```
max_depths = list(range(1,9,1))
validation_accs = []
validation_f1 = []
dt_models = []
for max_depth in max_depths :
    clf_dt = DecisionTreeClassifier(criterion = 'gini', max_depth = max_depth)
    clf_dt.fit(X_train, y_train)
    dt_models.append(clf_dt)
    y_valid_pred = clf_dt.predict(X_valid)
    validation_accs.append(accuracy_score(y_valid, y_valid_pred))
    validation_f1.append(f1_score(y_valid, y_valid_pred, pos_label = 1))
```

In [19]:
```
fig, ax = plt.subplots(figsize = (7,7))
ax.plot(max_depths, validation_accs, color = 'red', marker='o', label = 'validation a
ax.plot(max_depths, validation_f1, color = 'blue', marker='s', label = 'validation f1
ax.legend(loc = 'best')
plt.show(fig)
```



아래부터는 5-CV GridSearch를 통해 최적 파라미터를 탐색하는 과정입니다.

GridSearch

In [42]:
```
base = DecisionTreeClassifier()
opt = {'criterion' : ['entropy'], 'max_depth' : [1,2,3,4,5]}
cv = GridSearchCV(estimator = base, param_grid = opt, cv = 5)
cv_fitted = cv.fit(X_train, y_train)
```

In [43]:
```
cv_fitted.best_params_
```

Out[43]:
```
{'criterion': 'entropy', 'max_depth': 5}
```

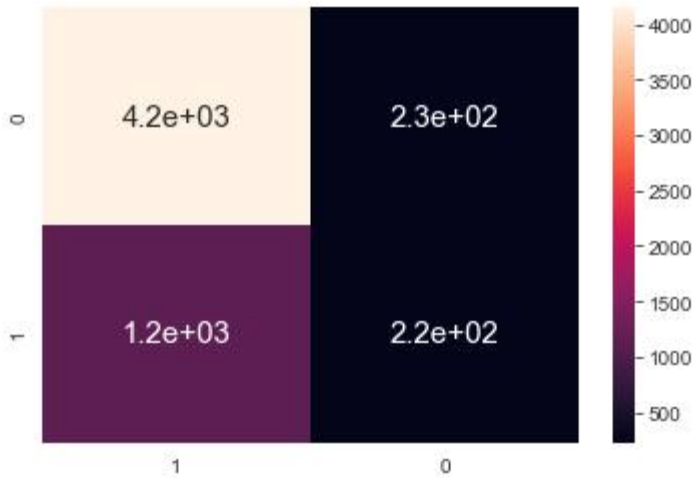
In [128]:
```
y_train_pred = cv.predict(X_train)
y_valid_pred = cv.predict(X_valid)

print('- Accuracy (Train) : {:.4}'.format(accuracy_score(y_train, y_train_pred)))
print('- Accuracy (Validation) : {:.4}'.format(accuracy_score(y_valid, y_valid_pred)))

print('- F1 score (Train) : {:.4}'.format(f1_score(y_train, y_train_pred)))
print('- F1 score (Validation) : {:.4}'.format(f1_score(y_valid, y_valid_pred)))

print('- precision score(Train) : {:.4}'.format(precision_score(y_train, y_train_pred)
print('- precision score(Validation) : {:.4}'.format(precision_score(y_valid, y_valid_

print('- recall score (Train) : {:.4}'.format(recall_score(y_train, y_train_pred)))
print('- recall score (Validation) : {:.4}'.format(recall_score(y_valid, y_valid_pred)
```

```
- Accuracy (Train) : 0.7663
- Accuracy (Validation) : 0.7677
- F1 score (Train) : 0.1554
- F1 score (Validation) : 0.1667
- precision score(Train) : 0.5756
- precision score(Validation) : 0.5912
- recall score (Train) : 0.08982
- recall score (Validation) : 0.09701
```

최적의 모델에 대한 정확도와 F1 Score 등등 을 구합니다.

```
In [57]: cm_test = confusion_matrix(y_test, y_test_pred)
         cm_test

Out[57]: array([[4152,  227],
                [1154,  224]], dtype=int64)
```

```
In [54]: cm_test = confusion_matrix(y_test, y_test_pred)
         cm_test = pd.DataFrame(cm_test, columns = [1,0])
         sns.heatmap(data=cm_test, annot=True, annot_kws={'size':15})

Out[54]: <AxesSubplot:>
```



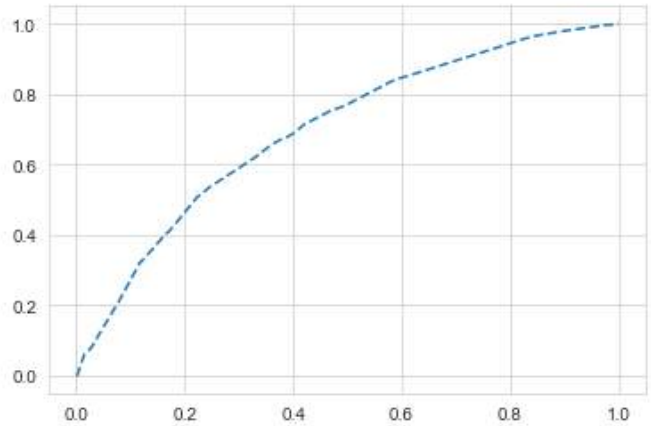
위와 같이 confusion maxtrix도 그려주었습니다.

```
In [58]: # Best Model에 대한 규칙(시각화)
         graph = Source(export_graphviz(clf_dt,
                                        out_file=None,
                                        filled=True,
                                        leaves_parallel=False))
         Image(graph.pipe(format='png'))

Out[58]:
```

```
In [67]: # 변수의 중요도
         imp = dt_models[best_model_idx].feature_importances_
         imp

Out[67]: array([0.        , 0.        , 0.        , 0.12846532, 0.        ,
                0.        , 0.        , 0.        , 0.        , 0.05382076,
                0.        , 0.03752959, 0.        , 0.        , 0.        ,
                0.78018433])
```

Best_model에 대한 시각화와 변수 중요도도 구해보았습니다.

```
In [135]: y_score_train =cv.predict_proba(X_train)
          y_score_valid =cv.predict_proba(X_valid)

          # Show ROC Curve
          fpr, tpr, thresholds = roc_curve(y_valid, y_score_valid[:,1], pos_label = 1)
          plt.plot(fpr, tpr, linestyle='--')
Out[135]: [<matplotlib.lines.Line2D at 0x19e6a84d910>]
```



ROC Curve도 그려줍니다.

```
In [136]: print(auc(fpr,tpr))
          0.6983410670662034
```

Auc score 값을 계산해줍니다.

```
In [137]: alanced_accuracy_score (Test) : {:.4}'.format(balanced_accuracy_score(y_test,pred_y)))
          - balanced_accuracy_score (Test) : 0.5
```

균형정확도 계산도 해주었습니다.

Bagging

Bagging

In [38]:
```
# Bagging
# X와 y 분리
y = data['TARGET'].values
X = data.drop('TARGET',axis = 1)
xcolumns = X.columns.values
X = X.values
```

In [22]:
```
# test data와 train data 분리
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=20
```

In [23]:
```
tree = DecisionTreeRegressor()
tree.fit(X_train, y_train)
y_tree = tree.predict(X_test)
```

In [24]:
```
treeBagging = BaggingRegressor(base_estimator = DecisionTreeRegressor(),
                               n_estimators = 100)
treeBagging.fit(X_train, y_train)
y_treeBagging = treeBagging.predict(X_test)
```

In [25]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_lr = lr.predict(X_test)
```

In [26]:
```
lrBagging = BaggingRegressor(base_estimator = LinearRegression(),
                             n_estimators = 100)
lrBagging.fit(X_train, y_train)
y_lrBagging = lrBagging.predict(X_test)
```

데이터를 분리하고 학습시켜 줍니다.

Best_estimator는 샘플링한 데이터를 학습시킬 때 어떤 모델로 학습시킬 것인가를 결정하는 것이고, n_estimator는 몇 개의 데이터를 만들것인지 정하는 것입니다.

기본 deicision tree, tree기반의 bagging, 기본 linear regresssion, Linear regression 기반의 bagging 이렇게 총 4개의 데이터를 만들어주었습니다.

Evaluation

In [27]:
```
mse_tree = mean_squared_error(y_test, y_tree)
mse_treeBagging = mean_squared_error(y_test, y_treeBagging)
mse_lr = mean_squared_error(y_test, y_lr)
mse_lrBagging = mean_squared_error(y_test, y_lrBagging)

r2_tree = r2_score(y_test, y_tree)
r2_treeBagging = r2_score(y_test, y_treeBagging)
r2_lr = r2_score(y_test, y_lr)
r2_lrBagging = r2_score(y_test, y_lrBagging)
```

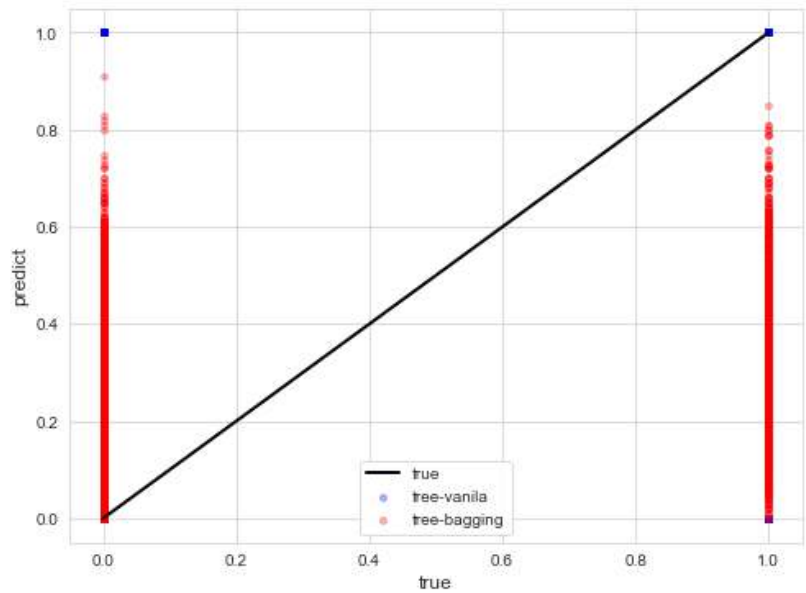
In [28]:
```
print('[Tree - Vanila] R-square = {:.2f} | MSE = {:.2f}'.format(r2_tree, mse_tree))
print('[Tree - Bagging] R-square = {:.2f} | MSE = {:.2f}'.format(r2_treeBagging, mse_t
print('[Linear - Vanila] R-square = {:.2f} | MSE = {:.2f}'.format(r2_lr, mse_lr))
print('[Linear - Bagging] R-square = {:.2f} | MSE = {:.2f}'.format(r2_lrBagging, mse_l
```

```
[Tree - Vanila] R-square = -0.86 | MSE = 0.34
[Tree - Bagging] R-square = 0.07 | MSE = 0.17
[Linear - Vanila] R-square = 0.10 | MSE = 0.16
[Linear - Bagging] R-square = 0.10 | MSE = 0.16
```

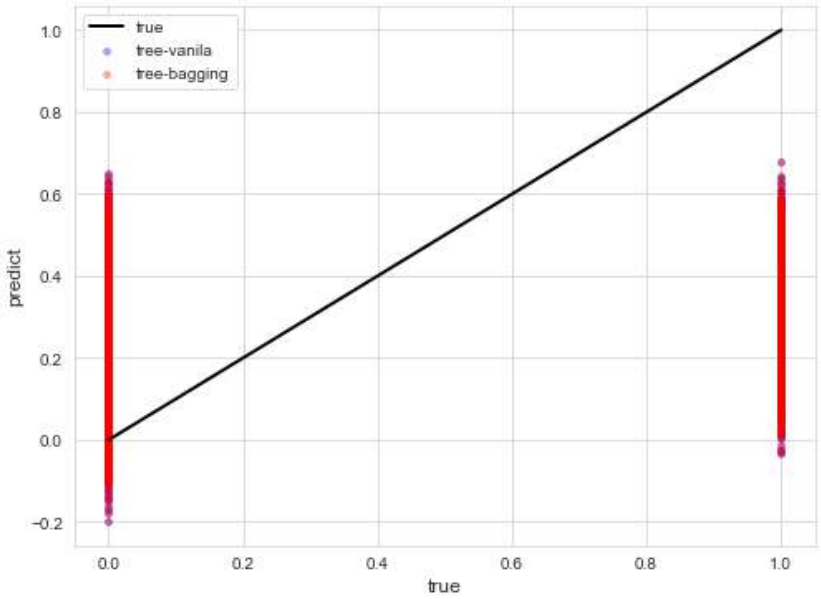
그리고 각각 모델의 R-squre과 MSE를 구해주었습니다.
다음은 학습 결과를 visualization한 결과입니다.

Visualization

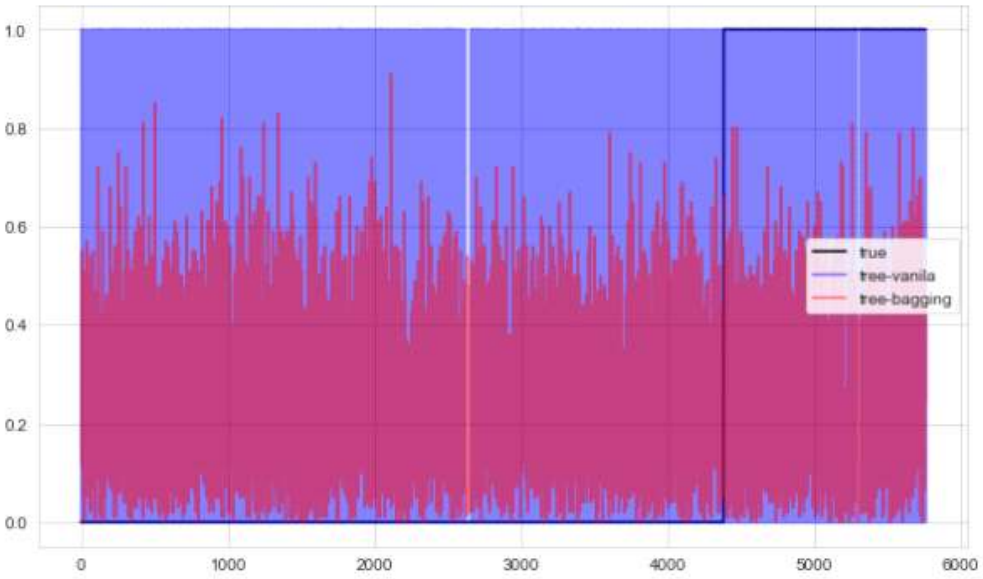
In [31]:
```
plt.figure(figsize = (8,6))
plt.scatter(y_test, y_tree, s = 15, alpha = 0.3, color = 'blue', label = 'tree-vanila
plt.scatter(y_test, y_treeBagging, s = 15, alpha = 0.3, color = 'red', label = 'tree-
plt.plot(y_test, y_test, alpha = 1.0, lw = 2, color = 'black', label = 'true')
plt.xlabel('true', fontsize = 12)
plt.ylabel('predict', fontsize = 12)
plt.legend()
plt.show()
```



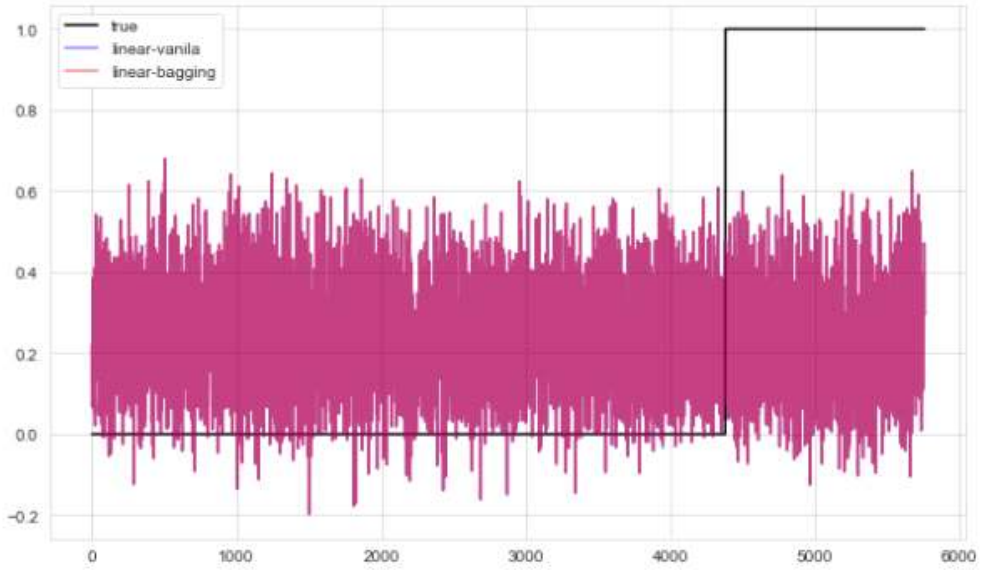
In [32]:
```
plt.figure(figsize = (8,6))
plt.scatter(y_test, y_lr, s = 15, alpha = 0.3, color = 'blue', label = 'tree-vanila')
plt.scatter(y_test, y_lrBagging, s = 15, alpha = 0.3, color = 'red', label = 'tree-ba
plt.plot(y_test, y_test, alpha = 1.0, lw = 2, color = 'black', label = 'true')
plt.xlabel('true', fontsize = 12)
plt.ylabel('predict', fontsize = 12)
plt.legend()
plt.show()
```



In [36]:
```
idx = y_test.argsort()
plt.figure(figsize = (10,6))
plt.plot(y_test[idx], color = 'black', alpha = 1.0, label = 'true')
plt.plot(y_tree[idx], color = 'blue', alpha = 0.5, label = 'tree-vanila')
plt.plot(y_treeBagging[idx], color = 'red', alpha = 0.5, label = 'tree-bagging')
plt.legend()
plt.show()
```



In [40]:
```
plt.figure(figsize = (10,6))
plt.plot(y_test[idx], color = 'black', alpha = 1.0, label = 'true')
plt.plot(y_lr[idx], color = 'blue', alpha = 0.5, label = 'linear-vanila')
plt.plot(y_lrBagging[idx], color = 'red', alpha = 0.5, label = 'linear-bagging')
plt.legend()
plt.show()
```



생각보다 결과가 잘 안나오는 것을 확인할 수 있습니다.
수업 때 실습한 자료는 이산형이었는데 이번 실습에서 다룬 데이터들은 범주형 변수들이 많아서 이런 결과가 나온 것 같습니다.

GridSearch

In [89]:
```
# Bagging
bg_clf = BaggingClassifier(random_state=20210619)
bg_clf.fit(X_train, y_train)
bg_pred = bg_clf.predict(X_test)
bg_accuracy = accuracy_score(y_test, bg_pred)

param = {'n_estimators' : [100, 500]}
bg_cv = GridSearchCV(bg_clf, param_grid=param, cv=5)
bg_cv.fit(X_train, y_train)

print('최적 하이퍼 파라미터 : ₩n',bg_cv.best_params_)
print('최고 예측 정확도 : {:.4f}'.format(bg_cv.best_score_))
```

```
최적 하이퍼 파라미터 :
 {'n_estimators': 500}
최고 예측 정확도 : 0.7584
```

5-CV Grid Search를 통해 최적 파라미터를 탐색하였습니다.
N_estimator가 500일 때 최적이었습니다.
이 예측 모델을 사용하였을 때 정확도는 75.84%가 나왔습니다.

Random Forest

Random Forest

In [34]:
```
y = data['TARGET'].values
X = data.drop('TARGET',axis = 1)
xcolumns = X.columns.values
X = X.values

# test data와 train data 분리
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=20
```

In [90]:
```
# Grid Search
rf = RandomForestClassifier(random_state=2021)
param_grid = {'n_estimators' : [100, 200],
              'criterion' : ['gini'],
              'max_depth':[None, 6]}
rf_cv = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5)
rf_cv.fit(X_train, y_train)
```

Out[90]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=2021),
             param_grid={'criterion': ['gini'], 'max_depth': [None, 6],
                         'n_estimators': [100, 200]})
```

Random Forest 부분입니다.
데이터를 분리 후 RandomForestClassifier을 통해 학습시켜주었습니다.

In [169]:
```
rf_cv.best_estimator_
```

Out[169]:
```
RandomForestClassifier(max_depth=6, n_estimators=200, random_state=2021)
```

In [172]:
```
y_pred = rf_cv.best_estimator_.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[172]:
```
array([[4350,   29],
       [1336,   42]], dtype=int64)
```

In [171]:
```
acc = accuracy_score(y_test, y_pred)
print('[Accuracy] : {:.2f}%'.format(acc*100))
```

```
[Accuracy] : 76.29%
```

학습 시키고 나서의 confusion matrix와 정확도를 출력시켰습니다.
정확도는 약 76.29%가 나오는 것을 확인할 수 있었습니다.

Adaboost

AdaBoost

In [85]:
```
# AdaBoost
ada_clf = AdaBoostClassifier(random_state=20210619)
ada_clf.fit(X_train, y_train)
ada_pred = ada_clf.predict(X_test)
ada_accuracy = accuracy_score(y_test, ada_pred)

param = {'n_estimators' : [100, 500],
         'learning_rate' : [0.05, 0.1]}
ada_cv = GridSearchCV(ada_clf, param_grid=param, cv=5)
ada_cv.fit(X_train, y_train)

print('최적 하이퍼 파라미터 : ₩n',ada_cv.best_params_)
print('최고 예측 정확도 : {:.4f}'.format(ada_cv.best_score_))
```

```
최적 하이퍼 파라미터 :
 {'learning_rate': 0.05, 'n_estimators': 500}
최고 예측 정확도 : 0.7670
```

In [93]:
```
y_pred = ada_cv.best_estimator_.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[93]:
```
array([[4283,   96],
       [1250,  128]], dtype=int64)
```

In [94]:
```
acc = accuracy_score(y_test, y_pred)
print('[Accuracy] : {:.2f}%'.format(acc*100))
```

```
[Accuracy] : 76.62%
```

Adaboost 또한 마찬가지로 데이터 분리 후 AdaBoostClassifier를 통해 학습 시켜준 뒤에, 최적 하이퍼 파라미터를 구하였습니다. 최적의 모델을 사용한 뒤에 예측 정확도는 76.62%가 나왔습니다.

Gradient Boosting

Gradient Boosting

In [140]:
```
y = data['TARGET'].values
X = data.drop('TARGET',axis = 1)
X = X.values

# test data와 train data 분리
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=20
```

In [141]:
```
gb = GradientBoostingRegressor(learning_rate = 0.1, n_estimators = 50)
```

In [142]:
```
xgbr = xgb.XGBRegressor(learning_rate = 0.1, n_estimators = 50, max_depth = 3, reg_la
```

In [143]:
```
trained_xgb = xgbr.fit(X_train, y_train)
trained_gb = gb.fit(X_train, y_train)
```

In [144]:
```
trained_pred_xgb = trained_xgb.predict(X_train)
trained_pred_gb = trained_gb.predict(X_train)
test_pred_xgb = trained_xgb.predict(X_test)
test_pred_gb = trained_gb.predict(X_test)
```

In [145]:
```
trained_mse_xgb = mean_squared_error(trained_pred_xgb, y_train)
trained_mse_gb = mean_squared_error(trained_pred_gb, y_train)
trained_r2_xgb = r2_score(trained_pred_xgb, y_train)
trained_r2_gb = r2_score(trained_pred_gb, y_train)

test_mse_xgb = mean_squared_error(trained_pred_xgb, y_train)
test_mse_gb = mean_squared_error(trained_pred_gb, y_train)
test_r2_xgb = r2_score(test_pred_xgb, y_test)
test_r2_gb = r2_score(test_pred_gb, y_test)

print('Train MSE : {:.3f} -> {:.3f}'.format(trained_mse_gb, trained_mse_xgb))
print('Train r2 : {:.3f} -> {:.3f}'.format(trained_r2_gb, trained_r2_xgb))
print('Train MSE : {:.3f} -> {:.3f}'.format(test_mse_gb, test_mse_xgb))
print('Train r2 : {:.3f} -> {:.3f}'.format(test_r2_gb, test_r2_xgb))
```

```
Train MSE : 0.159 -> 0.159
Train r2 : -8.170 -> -8.209
Train MSE : 0.159 -> 0.159
Train r2 : -8.383 -> -8.481
```

데이터를 분리하고 lr을 0.1, n_estimator를 50, max_depth를 3, reg_lambda를 0.1로 설정해 주었습니다. 그 다음 fitting을 통해 학습시켜주고 MSE의 변화율을 살펴보았습니다.

In [146]:
```
# Gradient Boosting
y = data['TARGET'].values
X = data.drop('TARGET',axis = 1)
xcolumns = X.columns.values
X = X.values

# test data와 train data 분리
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=20
```

In [154]:
```
gb_clf = GradientBoostingClassifier(random_state=20210619)
gb_clf.fit(X_train, y_train)
gb_pred = gb_clf.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_pred)
```

In [157]:
```
param = {'n_estimators' : [100, 500],
         'learning_rate' : [0.05, 0.1]}
grid_cv = GridSearchCV(gb_clf, param_grid=param, cv=5)
grid_cv.fit(X_train, y_train)
print('최적 하이퍼 파라미터 : ₩n',grid_cv.best_params_)
print('최고 예측 정확도 : {:.4f}'.format(grid_cv.best_score_))
```

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits
최적 하이퍼 파라미터 :
 {'learning_rate': 0.05, 'n_estimators': 100}
최고 예측 정확도 : 0.7651
```

이번에는 최적의 파라미터를 찾아서 학습시켜보았습니다.
최적의 하이퍼 파라미터는 lr = 0.05, n_estimator는 100이었습니다.
이 모델을 사용했을 때, 최고 예측 정확도는 76.51%가 나왔습니다.

결론

수업시간 때 다루었던 데이터는 출력 변수가 이산형이었지만, 실습 과제에서 다룬 데이터는 이진 범주형이라라서 생각했던 출력값과 조금 다르게 나와서 당황하였습니다.
5-CV GridSearch를 통해 최적 파라미터로 학습한 결과를 비교하자면,
Decision Tree : 76.63%
Bagging : 75.84%
Random Forest : 76.29%
Adaboost : 76.62%
Gradient Boosting : 76.51%
Decision Tree가 제일 높은 결과가 나왔지만 큰 차이는 없었습니다.
평균 약 76%의 정확도를 보이고 있습니다.
80%이상의 정확도가 나오지 않는게 조금 아쉽긴 했습니다.