5_gradient_boosting_exercise

May 31, 2019

In [ ]:
```
import hourse_price_preprocessor
```

In [ ]:
```
train_dataset_dir = "./house_price/train.csv"
test_dataset_dir = "./house_price/test.csv"
```

In [ ]:
```
import pandas as pd
import numpy as np

train = pd.read_csv("./house_price/train.csv" )
test = pd.read_csv("./house_price/test.csv" )

print(train.columns)
print(test.columns)
train.info()
```

In [ ]:
```
# 'Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature'
```

In [ ]:
```
# object data one-hot
```

In [ ]:
```
pd.DataFrame(train.isnull().sum())
```

In [ ]:
```
# 'LotFrontage', 'MasVnrArea', 'GarageYrBlt'
```

In [ ]:
```
# column null
```

In [ ]:
```
# x, y
```

In [ ]:
```
train_X, test_X, train_y, test_y = train_test_split(x, y, test_size=0.2)
```

In [ ]:
```
train_X.shape, test_X.shape, train_y.shape, test_y.shape
```

In [ ]:
```
train_X[:2]
```

In [ ]:
```
train_y[:10]
```

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import cross_val_score

import numpy as np
```

```
In [ ]: gbr = GradientBoostingRegressor(n_estimators=2000, subsample=0.5, \
                                       max_depth=2, learning_rate=0.55)
        lr = LinearRegression()

        # max_depth  estimator  - \
        tree

In [ ]: np.mean(cross_val_score(gbr, train_X, train_y, scoring="r2" , cv=5))

In [ ]: np.mean(cross_val_score(lr, train_X, train_y, scoring="r2" , cv=5))

In [ ]: np.mean(cross_val_score(lr, train_X, train_y, cv=5))

In [ ]: def rmse(predictions, targets):
            return np.sqrt(((predictions - targets) ** 2).mean())

In [ ]: from sklearn.model_selection import train_test_split

In [ ]: gbr = GradientBoostingRegressor(
                max_depth=2, n_estimators=5000, subsample=0.5,\
            learning_rate=0.05)
        X_train, X_val, y_train, y_val = \
                train_test_split(train_X, train_y, test_size=0.3)
        # X_train = train_X
        # X_test = test_X
        # y_train = train_y
        # y_test = test_y
        gbr.fit(X_train, y_train)
        errors_val = [rmse(y_val, y_pred) for y_pred in \
                      gbr.staged_predict(X_val)]
        # gbr.staged_predict - estimator  , predict
        errors_train = [rmse(y_train, y_pred) for y_pred in \
                        gbr.staged_predict(X_train)]
        #errors_val errors_train  overfitting
        x_axis = list(range(len(errors_val)))

        lr = LinearRegression()
        lr.fit(X_train, y_train)

In [ ]: import matplotlib.pyplot as plt
        import numpy as np


        ax = plt.subplot(111)
        plt.plot(x_axis, errors_val, label="validation_score")
        plt.plot(x_axis, errors_train, label="train_score")

        leg = plt.legend(loc='best', ncol=2, mode="expand", \
                         shadow=True, fancybox=True)
```

```
        leg.get_frame().set_alpha(0.5)


        plt.show()
        bst_n_estimators = np.argmin(errors_val)
        print(bst_n_estimators, errors_val[bst_n_estimators])
        print(rmse(y_val, lr.predict(X_val)))
```

In [ ]:
```
gbr = GradientBoostingRegressor(
        max_depth=2, n_estimators=500, subsample=0.5,\
    learning_rate=0.05)
X_train, X_val, y_train, y_val = \
    train_test_split(train_X, train_y, test_size=0.3)
gbr.fit(X_train, y_train)
print(gbr.score(X_train, y_train))
print(gbr.score(X_val, y_val))
```

In [ ]:
```
gbr_best = GradientBoostingRegressor(max_depth=2, \
                subsample=0.5,learning_rate=0.05, n_estimators=500)
gbr_best.fit(train_X, train_y)
```

In [ ]:
```
test_y
```

In [ ]:
```
id_value=test_y
sales_price = gbr_best.predict(test_X)
```

In [ ]:
```
result = np.vstack([id_value, sales_price]).T

result
```

In [ ]:
```
from pandas import DataFrame

DataFrame(result, dtype=int, columns=["Id", "SalePrice"]).\
            set_index("Id").to_csv("house_price_result.csv")
# DataFrame(result, columns=["Id", "SalePrice"]).s\
#         et_index("Id").to_csv("house_price_result.csv")
```

In [ ]:
```
from sklearn.cross_validation import ShuffleSplit
from sklearn.grid_search import GridSearchCV

param_grid={'n_estimators':[500, 1000, 2000],
                'learning_rate': [0.1, 0.05], #0.02, 0.01],
                'subsample' : [0.4,0.5],#,0.6,0.7,0.8],
                'max_depth':[2, 4], #6 ,8],
                'min_samples_leaf':[3, 5],#,9,15],
                'max_features':[1.0, 0.3]#, 0.1]
            }
n_jobs=-7
```

```
        estimator = GradientBoostingRegressor(warm_start=True)
        cv = ShuffleSplit(X_train.shape[0], n_iter=5, test_size=0.2)
        classifier = GridSearchCV(estimator=estimator, cv=cv, \
                                  param_grid=param_grid, n_jobs=n_jobs, verbose=1)

        classifier.fit(train_X, train_y)
        print (classifier.best_estimator_)
```

In [ ]:
```
print(rmse(y_val, classifier.best_estimator_.predict(X_val)))
```

In [ ]:
```
print(rmse(y_val, classifier.predict(X_val)))
```

In [ ]:

In [ ]:
```
final_estimator = classifier.best_estimator_
final_estimator.fit(train_X, train_y)
```

In [ ]:
```
print(rmse(y_val, final_estimator.predict(X_val)))
```

In [ ]:
```
print(final_estimator.score(X_train, y_train))
```

In [ ]:
```
print(final_estimator.score(X_val, y_val))
```

In [ ]:
```
id_value=test_y
sales_price = final_estimator.predict(test_X)
result = np.vstack([id_value, sales_price]).T
submission_df =DataFrame(result, columns=["Id", "SalePrice"]).set_index("Id")
submission_df.index = submission_df.index.astype(int)
submission_df.to_csv("house_price_result_grid_search.csv")
```

In [ ]: