Booklet3Code

August 8, 2020

1 Booklet 3 Ensemblemethoden

[ ]:
```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
 ↪AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import matplotlib
```

1.1 Univariat Grid Search

[ ]:
```
params_bagging = [{'n_estimators': [10, 50, 100]},
                   {"base_estimator__criterion": ["gini", "entropy"]},
                   {"base_estimator__max_depth": [None, 3,5,10, 15, 20, 25,␣
 ↪50]},
                   {"base_estimator__min_samples_split":␣
 ↪[2,5,10,20,30,40]},#The minimum number of samples required to split an␣
 ↪internal node:
                   {"base_estimator__min_samples_leaf": [1,2,5,10,20,40]},#The␣
 ↪minimum number of samples required to be at a leaf node
                   {"base_estimator__min_weight_fraction_leaf": [0, 0.2, 0.4, 0.
 ↪5]},#The minimum weighted fraction of the sum total of weights (of all the␣
 ↪input samples) required to be at a leaf node.
                   {"base_estimator__max_features": [5,10,15,20, 25, None]},
                   {"base_estimator__max_leaf_nodes": [None, 2␣
 ↪,10,100,150,200,300,500]},
                   {"base_estimator__min_impurity_decrease": [0.0, 0.1, 0.2, 0.
 ↪5, 0.8, 1, 2, 3]},
                   {"base_estimator__min_impurity_split": [0.0, 0.1, 0.2, 0.5,␣
 ↪0.8, 1, 2, 3]} #dericated option
                  ]

params_adaboost = [{'n_estimators': [10, 50, 100]},
                   {'learning_rate': [0.1, 0.5, 1]},
                   {"base_estimator__criterion": ["gini", "entropy"]},
                   #{"base_estimator__max_depth": [None, 3,5,10, 15, 20, 25,␣
 ↪50]},
```

```
                    {"base_estimator__min_samples_split":␣
↪[2,5,10,20,30,40]},#The minimum number of samples required to split an␣
↪internal node:
                    {"base_estimator__min_samples_leaf": [1,2,5,10,20,40]},#The␣
↪minimum number of samples required to be at a leaf node
                    {"base_estimator__min_weight_fraction_leaf": [0, 0.2, 0.4, 0.
↪5]},#The minimum weighted fraction of the sum total of weights (of all the␣
↪input samples) required to be at a leaf node.
                    {"base_estimator__max_features": [1,2,10,15,20, None]},
                    {"base_estimator__max_leaf_nodes":␣
↪[None,2,5,10,100,150,200]},
                    {"base_estimator__min_impurity_decrease": [0.0, 0.1, 0.2, 0.
↪5, 0.8, 1, 2, 3]},
                    {"base_estimator__min_impurity_split": [0.0, 0.1, 0.2, 0.5,␣
↪0.8, 1, 2, 3]} #dericated option
                   ]

params_random_forest = [{'n_estimators': [10, 50, 100, 200, 500]},
                    {"criterion": ["gini", "entropy"]},
                    {"max_depth": [None, 3,5,10, 15, 20, 25, 50]},
                    #{"min_samples_split": [2,5,10,20,30,40]},#The minimum␣
↪number of samples required to split an internal node:
                    {"min_samples_leaf": [1,2,5,10,20,40]},#The minimum number␣
↪of samples required to be at a leaf node
                    {"min_weight_fraction_leaf": [0, 0.2, 0.4, 0.5]},#The␣
↪minimum weighted fraction of the sum total of weights (of all the input␣
↪samples) required to be at a leaf node.
                    {"max_features": [5,10,15,20, 25, None]},
                    {"max_leaf_nodes": [None, 2 ,10,100,150,200,300,500]},
                    {"min_impurity_decrease": [0.0, 0.1, 0.2, 0.5, 0.8, 1, 2,␣
↪3]},
                    {"min_impurity_split": [0.0, 0.1, 0.2, 0.5, 0.8, 1, 2, 3]}␣
↪#dericated option
                   ]

params_tree = [{"criterion": ["gini", "entropy"]},
                    #{"max_depth": [None, 3,5,10, 15, 20, 25, 50]},
                    {"min_samples_split": [2,5,10,20,30,40]},#The minimum number␣
↪of samples required to split an internal node:
                    {"min_samples_leaf": [1,2,5,10,20,40]},#The minimum number␣
↪of samples required to be at a leaf node
                    {"min_weight_fraction_leaf": [0, 0.2, 0.4, 0.5]},#The␣
↪minimum weighted fraction of the sum total of weights (of all the input␣
↪samples) required to be at a leaf node.
                    {"max_features": [5,10,15,20, 25, None]},
                    {"max_leaf_nodes": [None, 2 ,10,100,150,200,300,500]},
```

```
                   {"min_impurity_decrease": [0.0, 0.1, 0.2, 0.5, 0.8, 1, 2,␣
↪3]},
                   {"min_impurity_split": [0.0, 0.1, 0.2, 0.5, 0.8, 1, 2, 3]}␣
↪#dericated option
                  ]
```

[ ]:
```
#looping through grid ourselfes for better interpretation of outputs.
for param in params_random_forest:

    bagging_classifier = BaggingClassifier(
        base_estimator=DecisionTreeClassifier(),
        bootstrap=True, #replace training samples
        n_jobs=-1, #use all available cores
        random_state=42,
  #      n_estimators=100
    )

    bagging_gs = GridSearchCV(bagging_classifier, param, cv=10)
    bagging_gs.fit(X_train, y_train)
    print(param)
    print(bagging_gs.best_params_)
    print(bagging_gs.best_score_)
    print(bagging_gs.cv_results_['mean_test_score'])
    print(" ")
```

[ ]:
```
start_time = time.time()

for param in params_random_forest:

    adaboost_classifier = AdaBoostClassifier(
        base_estimator=DecisionTreeClassifier(
        max_depth=5,
        max_features=25,

            min_impurity_decrease=0.0005,
            min_samples_split= 10
        ),
        random_state=42,

    )

    adaboost_gs = GridSearchCV(adaboost_classifier, param, cv=10)
    adaboost_gs.fit(X_train, y_train)
    print(param)
    print(adaboost_gs.best_params_)
    print(adaboost_gs.best_score_)
    print(adaboost_gs.cv_results_['mean_test_score'])
```

```
    print(" ")

elapsed_time = time.time() - start_time
print(elapsed_time)
```

[ ]:
```
# multivariat parameter space
params_tree = [{"max_depth": [None,3,5,7],
     "min_samples_leaf": [50, 100, 200]
              }]

params_random_forest = [{
    "base_estimator__max_depth": [None,3, 5, 10],
    "base_estimator__min_samples_split": [5, 10],
    "base_estimator__max_features": [None, 5, 10, 25],
    "base_estimator__min_impurity_decrease": [0.0001, 0.0005]
              }]
```

[ ]:
```
start_time = time.time()

for param in params_random_forest:

    tree_classifier = DecisionTreeClassifier(
        random_state=42,
    )


    tree_gs = GridSearchCV(tree_classifier, params_random_forest, cv=10)
    tree_gs.fit(X_train, y_train)
    print(param)
    print(tree_gs.best_params_)
    print(tree_gs.best_score_)
    print(tree_gs.cv_results_['mean_test_score'])
    print(" ")

elapsed_time = time.time() - start_time
print(elapsed_time)
```

[ ]:
```
start_time = time.time()

for param in params_random_forest:

    random_forest_classifier = RandomForestClassifier(
        bootstrap=True, #replace training samples
        random_state=42,
        n_jobs=-1, #use all available cores,
        n_estimators=100,
    )
```

```
    random_forest_gs = GridSearchCV(random_forest_classifier, param, cv=10)
    random_forest_gs.fit(X_train, y_train)
    print(param)
    print(random_forest_gs.best_params_)
    print(random_forest_gs.best_score_)
    print(random_forest_gs.cv_results_['mean_test_score'])
    print(" ")

elapsed_time = time.time() - start_time
print(elapsed_time)
```

1.1.1 Optimized Bagging Classifier

[ ]:
```
# Ist das hier nicht im Prinzip das Gleiche wie ein Random Forest mit␣
 ↪splitter="random"?
start_time = time.time()

bagging_classifier = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(
     max_features=5,
        min_impurity_decrease=0.0001,
        min_samples_split=10
    ),
    bootstrap=True, #replace training samples
    n_jobs=-1, #use all available cores
    random_state=42,
    n_estimators=100
)
bagging_classifier.fit(X_train, y_train)

test_predictions = bagging_classifier.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions))
mean_absolute_error(y_test, test_predictions)

elapsed_time = time.time() - start_time
print(elapsed_time)
```

1.1.2 Optimized AdaBoost Classifier

[ ]:
```
start_time = time.time()
adaboost_classifier = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(
        max_depth=20,
        #max_features=25,
```

```
        #   min_impurity_decrease=0.0005,
         #  min_samples_split= 10
        ),
    n_estimators=100,
    random_state=42
)

adaboost_classifier.fit(X_train, y_train)

test_predictions = adaboost_classifier.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions))
mean_absolute_error(y_test, test_predictions)

elapsed_time = time.time() - start_time
print(elapsed_time)
```

1.1.3 Optimized Random Forest

[ ]:
```
random_forest_classifier = RandomForestClassifier(
    bootstrap=True, #replace training samples
    random_state=42,
    n_jobs=-1, #use all available cores,
    min_impurity_decrease=0.00005,
    min_samples_split=5,
    n_estimators=100
)

random_forest_classifier.fit(X_train, y_train)

test_predictions = random_forest_classifier.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions))
mean_absolute_error(y_test, test_predictions)

# Feature importances with random forest
for name, score in zip(X_train.columns, random_forest_classifier.
 ↪feature_importances_):
    print(name, score)
```

1.1.4 Optimized Tree

[ ]:
```
tree_classifier = DecisionTreeClassifier(
    random_state=42,
    min_impurity_decrease=0.0002,
    min_samples_split=6
    )
```

```
tree_classifier.fit(X_train, y_train)

test_predictions = tree_classifier.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions))
mean_absolute_error(y_test, test_predictions)
```

1.1.5 Visualisation of GridSearch results

[ ]:
```
parameter = {"max_depth": [3,5,10, 15, 20, 25, 50,100]}
#parameter = {"min_impurity_decrease": [0.0001, 0.001, 0.003, 0.005, 0.01, 0.
↪05, 0.1, 0.2]}
parameter_ = {"base_estimator__max_depth": [3,5,10, 15, 20, 25, 50,100]}

## Tree
tree_classifier = DecisionTreeClassifier(
    random_state=42,
)

tree_gs = GridSearchCV(tree_classifier, parameter, cv=10,␣
↪return_train_score=True)
tree_gs.fit(X_train, y_train)

results_tree = tree_gs.cv_results_
test_scores_tree = results_tree['mean_test_score']
train_scores_tree = results_tree['mean_train_score']


random_forest_classifier = RandomForestClassifier(
    bootstrap=True, #replace training samples
    random_state=42,
    n_jobs=-1, #use all available cores,
)

## Forest
random_forest_gs = GridSearchCV(random_forest_classifier, parameter, cv=10,␣
↪return_train_score=True)
random_forest_gs.fit(X_train, y_train)

results_forest = random_forest_gs.cv_results_
test_scores_forest = results_forest['mean_test_score']
train_scores_forest = results_forest['mean_train_score']

## bagging
bagging_classifier = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    bootstrap=True, #replace training samples
```

```
    n_jobs=-1, #use all available cores
    random_state=42
)

bagging_gs = GridSearchCV(bagging_classifier, parameter_, cv=10,␣
 ↪return_train_score=True)
bagging_gs.fit(X_train, y_train)

results_bagging = bagging_gs.cv_results_
test_scores_bagging = results_bagging['mean_test_score']
train_scores_bagging = results_bagging['mean_train_score']

## Adaboost
adaboost_classifier = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(),
    random_state=42
)

adaboost_gs = GridSearchCV(adaboost_classifier, parameter_, cv=10,␣
 ↪return_train_score=True)
adaboost_gs.fit(X_train, y_train)

results_adaboost = adaboost_gs.cv_results_
test_scores_adaboost = results_adaboost['mean_test_score']
train_scores_adaboost = results_adaboost['mean_train_score']


X_axis = np.array(results_tree['param_max_depth'].data, dtype=float)
```

[ ]:
```
results_tree
```

[ ]:
```
best_index_tree = np.nonzero(results_tree['rank_test_score'] == 1)[0][0]
best_score_tree = results_tree['mean_test_score'][best_index_tree]

best_index_adaboost = np.nonzero(results_adaboost['rank_test_score'] == 1)[0][0]
best_score_adaboost = results_adaboost['mean_test_score'][best_index_adaboost]

best_index_forest = np.nonzero(results_forest['rank_test_score'] == 1)[0][0]
best_score_forest = results_forest['mean_test_score'][best_index_forest]

best_index_bagging = np.nonzero(results_bagging['rank_test_score'] == 1)[0][0]
best_score_bagging = results_bagging['mean_test_score'][best_index_bagging]
```

[ ]:
```
matplotlib.rcParams['figure.figsize'] = (10.0, 8.0)
fig1, ax1 = plt.subplots()
```

```
ax1.plot(X_axis, train_scores_tree, color='blue', alpha=0.7, linestyle='dashed'␣
↪)
ax1.plot(X_axis, test_scores_tree, color='blue')

ax1.plot(X_axis, train_scores_forest, color = 'green', alpha=0.7,␣
↪linestyle='dashed')
ax1.plot(X_axis, test_scores_forest, color = 'green')

ax1.plot(X_axis, train_scores_adaboost, color = 'red', alpha=0.7,␣
↪linestyle='dashed')
ax1.plot(X_axis, test_scores_adaboost, color = 'red')

ax1.plot(X_axis, train_scores_bagging, color = 'purple', alpha=0.7,␣
↪linestyle='dashed')
ax1.plot(X_axis, test_scores_bagging, color = 'purple')

ax1.set_xscale('log')
ax1.set_xticks([3,5,10,20,50,100])
ax1.get_xaxis().set_major_formatter(matplotlib.ticker.ScalarFormatter())
ax1.set_xlabel("max_depth", fontsize=14)
ax1.set_ylabel('Accuracy', fontsize=14)
plt.legend(['train tree','test tree', 'train forest', 'test forest','train␣
↪adaboost', 'test adaboost', 'train bagging', 'test bagging'], fontsize=11)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

ax1.plot([X_axis[best_index_tree], ] * 2, [0, best_score_tree], linewidth=1,
        linestyle='-.', color='blue', marker='x', markeredgewidth=3, ms=8)
ax1.annotate("%0.3f" % best_score_tree,
            (X_axis[best_index_tree], best_score_tree + 0.005),␣
↪backgroundcolor="w", fontsize=12)

ax1.plot([X_axis[best_index_adaboost], ] * 2, [0, best_score_adaboost],␣
↪linewidth=1,
        linestyle='-.', color='red', marker='x', markeredgewidth=3, ms=8)
ax1.annotate("%0.3f" % best_score_adaboost,
            (X_axis[best_index_adaboost], best_score_adaboost + 0.005),␣
↪backgroundcolor="w", fontsize=12)

ax1.plot([X_axis[best_index_forest], ] * 2, [0, best_score_forest], linewidth=1,
        linestyle='-.', color='green', marker='x', markeredgewidth=3, ms=8)
ax1.annotate("%0.3f" % best_score_forest,
            (X_axis[best_index_forest], best_score_forest + 0.005),␣
↪backgroundcolor="w", fontsize=12)
```

```
ax1.plot([X_axis[best_index_bagging], ] * 2, [0, best_score_bagging],␣
↪linewidth=1,
        linestyle='-.', color='purple', marker='x', markeredgewidth=3, ms=8)
ax1.annotate("%0.3f" % best_score_bagging,
            (X_axis[best_index_bagging], best_score_bagging + 0.005),␣
↪backgroundcolor="w", fontsize=12)

ax1.set_ylim(0.75,1)

plt.savefig('grid_search_max_depth.png')
```