03_random_forest_tuning

September 29, 2021

1 How to train and tune a random forest

[1]:
```python
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```python
%matplotlib inline

from pathlib import Path
import os, sys
import numpy as np

from numpy.random import choice
import pandas as pd
from scipy.stats import spearmanr

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.metrics import make_scorer
import joblib

import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```python
sys.path.insert(1, os.path.join(sys.path[0], '..'))
from utils import MultipleTimeSeriesCV
```

[4]:
```python
sns.set_style('white')
np.random.seed(seed=42)
```

[5]:
```python
results_path = Path('results', 'random_forest')
if not results_path.exists():
    results_path.mkdir(parents=True)
```

1.1 Get Data

[6]:
```
with pd.HDFStore('data.h5') as store:
    data =store['us/equities/monthly']
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 77788 entries, ('A', Timestamp('2006-12-31 00:00:00')) to ('ZION',
Timestamp('2017-11-30 00:00:00'))
Data columns (total 27 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   atr            77788 non-null  float64
 1   bb_down        77788 non-null  float64
 2   bb_high        77788 non-null  float64
 3   bb_low         77788 non-null  float64
 4   bb_mid         77788 non-null  float64
 5   bb_up          77788 non-null  float64
 6   macd           77788 non-null  float64
 7   natr           77788 non-null  float64
 8   rsi            77788 non-null  float64
 9   sector         77788 non-null  object
 10  return_1m      77788 non-null  float64
 11  return_3m      77788 non-null  float64
 12  return_6m      77788 non-null  float64
 13  return_12m     77788 non-null  float64
 14  beta           77788 non-null  float64
 15  SMB            77788 non-null  float64
 16  HML            77788 non-null  float64
 17  RMW            77788 non-null  float64
 18  CMA            77788 non-null  float64
 19  momentum_3     77788 non-null  float64
 20  momentum_6     77788 non-null  float64
 21  momentum_3_6   77788 non-null  float64
 22  momentum_12    77788 non-null  float64
 23  momentum_3_12  77788 non-null  float64
 24  year           77788 non-null  int64
 25  month          77788 non-null  int64
 26  target         77788 non-null  float64
dtypes: float64(24), int64(2), object(1)
memory usage: 16.4+ MB
```

[7]:
```
y = data.target
y_binary = (y > 0).astype(int)
X = pd.get_dummies(data.drop('target', axis=1))
```

1.2 Random Forests

1.2.1 Cross-validation parameters

[12]:
```
n_splits = 10
train_period_length = 60
test_period_length = 6
lookahead = 1

cv = MultipleTimeSeriesCV(n_splits=n_splits,
                          train_period_length=train_period_length,
                          test_period_length=test_period_length,
                          lookahead=lookahead)
```

1.2.2 Classifier

[13]:
```
rf_clf = RandomForestClassifier(n_estimators=100,             # default changed␣
 ↪from 10 to 100 in version 0.22
                                criterion='gini',
                                max_depth=None,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                min_weight_fraction_leaf=0.0,
                                max_features='auto',
                                max_leaf_nodes=None,
                                min_impurity_decrease=0.0,
                                min_impurity_split=None,
                                bootstrap=True,
                                oob_score=True,
                                n_jobs=-1,
                                random_state=42,
                                verbose=1)
```

Cross-Validation with default settings

[9]:
```
cv_score = cross_val_score(estimator=rf_clf,
                           X=X,
                           y=y_binary,
                           scoring='roc_auc',
                           cv=cv,
                           n_jobs=-1,
                           verbose=1)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:   14.0s remaining:    9.3s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   17.2s finished
```

[10]:
```
np.mean(cv_score)
```

[10]: 0.5236329094014294

1.2.3 Regression RF

[11]:
```
def rank_correl(y, y_pred):
    return spearmanr(y, y_pred)[0]

ic = make_scorer(rank_correl)
```

[12]:
```
rf_reg = RandomForestRegressor(n_estimators=100,
                                max_depth=None,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                min_weight_fraction_leaf=0.0,
                                max_features='auto',
                                max_leaf_nodes=None,
                                min_impurity_decrease=0.0,
                                min_impurity_split=None,
                                bootstrap=True,
                                oob_score=False,
                                n_jobs=-1,
                                random_state=None,
                                verbose=0,
                                warm_start=False)
```

[13]:
```
cv_score = cross_val_score(estimator=rf_reg,
                           X=X,
                           y=y,
                           scoring=ic,
                           cv=cv,
                           n_jobs=-1,
                           verbose=1)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:  1.3min remaining:   52.7s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  1.6min finished
```

[14]:
```
np.mean(cv_score)
```

[14]: 0.019866006883088493

1.3 Parameter Tuning

The key configuration parameters include the various hyperparameters for the individual decision trees introduced in the notebook decision_trees.

The following tables lists additional options for the two `RandomForest` classes:

Keyword	Default	Description
bootstrap	True	Bootstrap samples during training
n_estimators	10	# trees in the forest.
oob_score	False	Use out-of-bag samples to estimate the R2 on unseen data
warm_start	False	Reuse result of previous call to continue training and add more trees to the ensemble, otherwise, train a whole new forest

- The `bootstrap` parameter activates in the preceding bagging algorithm outline, which in turn enables the computation of the out-of-bag score (oob_score) that estimates the generalization accuracy using samples not included in the bootstrap sample used to train a given tree (see next section for detail).
- The `n_estimators` parameter defines the number of trees to be grown as part of the forest. Larger forests perform better, but also take more time to build. It is important to monitor the cross-validation error as a function of the number of base learners to identify when the marginal reduction of the prediction error declines and the cost of additional training begins to outweigh the benefits.
- The `max_features` parameter controls the size of the randomly selected feature subsets available when learning a new decision rule and split a node. A lower value reduces the correlation of the trees and, thus, the ensemble's variance, but may also increase the bias. Good starting values are `n_features` (the number of training features) for regression problems and `sqrt(n_features)` for classification problems, but will depend on the relationships among features and should be optimized using cross-validation.

Random forests are designed to contain deep fully-grown trees, which can be created using `max_depth=None` and `min_samples_split=2`. However, these values are not necessarily optimal, especially for high-dimensional data with many samples and, consequently, potentially very deep trees that can become very computationally-, and memory-, intensive.

The `RandomForest` class provided by sklearn support parallel training and prediction by setting the n_jobs parameter to the k number of jobs to run on different cores. The -1 value uses all available cores. The overhead of interprocess communication may limit the speedup from being linear so that k jobs may take more than 1/k the time of a single job. Nonetheless, the speedup is often quite significant for large forests or deep individual trees that may take a meaningful amount of time to train when the data is large, and split evaluation becomes costly.

As always, the best parameter configuration should be identified using cross-validation. The following steps illustrate the process:

1.3.1 Define Parameter Grid

[15]:
```
param_grid = {'n_estimators': [50, 100, 250],
              'max_depth': [5, 15, None],
              'min_samples_leaf': [5, 25, 100]}
```

1.3.2 Instantiate GridSearchCV

We will use 10-fold custom cross-validation and populate the parameter grid with values for the key configuration settings:

[16]:
```
gridsearch_clf = GridSearchCV(estimator=rf_clf,
                              param_grid=param_grid,
                              scoring='roc_auc',
                              n_jobs=-1,
                              cv=cv,
                              refit=True,
                              return_train_score=True,
                              verbose=1)
```

1.3.3 Fit Classifier

[18]:
```
gridsearch_clf.fit(X=X, y=y_binary)
```

```
Fitting 10 folds for each of 27 candidates, totalling 270 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 270 out of 270 | elapsed:  5.8min finished
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    1.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    3.5s finished
```

[18]:
```
GridSearchCV(cv=<utils.MultipleTimeSeriesCV object at 0x7fc2d2f90050>,
             estimator=RandomForestClassifier(n_jobs=-1, oob_score=True,
                                              random_state=42, verbose=1),
             n_jobs=-1,
             param_grid={'max_depth': [5, 15, None],
                         'min_samples_leaf': [5, 25, 100],
                         'n_estimators': [50, 100, 250]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

Persist Result

[19]:
```
joblib.dump(gridsearch_clf, results_path / 'gridsearch_clf.joblib')
```

[19]:
```
['gridsearch_clf.joblib']
```

[20]:
```
gridsearch_clf = joblib.load(results_path / 'gridsearch_clf.joblib')
```

[21]:
```
gridsearch_clf.best_params_
```

[21]:
```
{'max_depth': 15, 'min_samples_leaf': 5, 'n_estimators': 100}
```

[22]:
```
gridsearch_clf.best_score_
```

[22]:
```
0.5210305933900112
```

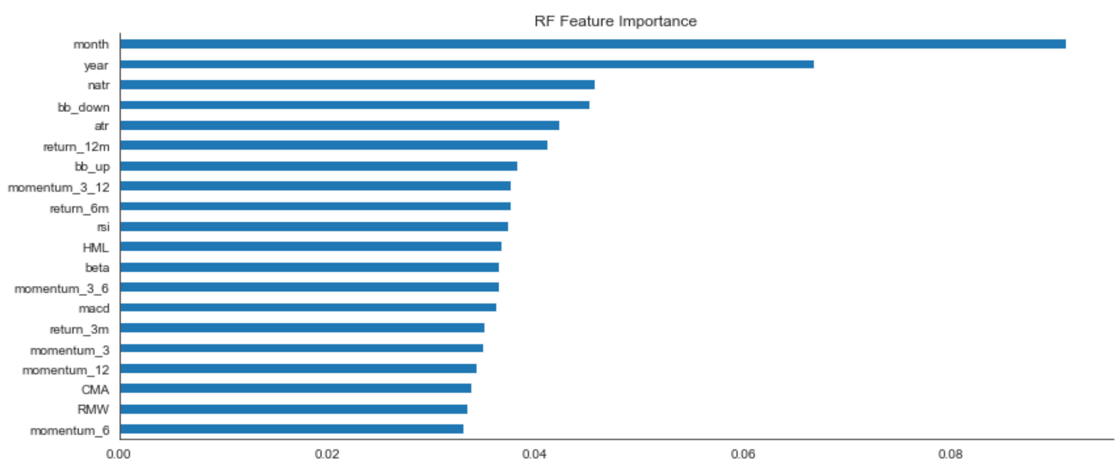
Feature Importance A random forest ensemble may contain hundreds of individual trees, but it is still possible to obtain an overall summary measure of feature importance from bagged models.

For a given feature, the importance score is the total reduction in the objective function's value, which results from splits based on this feature, averaged over all trees. Since the objective function takes into account how many features are affected by a split, this measure is implicitly a weighted average so that features used near the top of a tree will get higher scores due to the larger number of observations contained in the much smaller number of available nodes. By averaging over many trees grown in a randomized fashion, the feature importance estimate loses some variance and becomes more accurate.

The computation differs for classification and regression trees based on the different objectives used to learn the decision rules and is measured in terms of the mean square error for regression trees and the Gini index or entropy for classification trees.

`sklearn` further normalizes the feature-importance measure so that it sums up to 1. Feature importance thus computed is also used for feature selection as an alternative to the mutual information measures we saw in Chapter 6, The Machine Learning Process (see SelectFromModel in the sklearn.feature_selection module). In our example, the importance values for the top-20 features are as shown here:

[23]:
```python
fig, ax = plt.subplots(figsize=(12,5))
(pd.Series(gridsearch_clf.best_estimator_.feature_importances_,
           index=X.columns)
 .sort_values(ascending=False)
 .iloc[:20]
 .sort_values()
 .plot.barh(ax=ax, title='RF Feature Importance'))
sns.despine()
fig.tight_layout();
```



1.3.4 Fit Regressor

[24]:
```
gridsearch_reg = GridSearchCV(estimator=rf_reg,
                              param_grid=param_grid,
                              scoring=ic,
                              n_jobs=-1,
                              cv=cv,
                              refit=True,
                              return_train_score=True,
                              verbose=1)
```

[25]:
```
gs_reg = gridsearch_reg
```

[26]:
```
gridsearch_reg.fit(X=X, y=y)
```

```
Fitting 10 folds for each of 27 candidates, totalling 270 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 16.4min
[Parallel(n_jobs=-1)]: Done 270 out of 270 | elapsed: 27.8min finished
```

[26]:
```
GridSearchCV(cv=<utils.MultipleTimeSeriesCV object at 0x7fc2d2f90050>,
             estimator=RandomForestRegressor(n_jobs=-1), n_jobs=-1,
             param_grid={'max_depth': [5, 15, None],
                         'min_samples_leaf': [5, 25, 100],
                         'n_estimators': [50, 100, 250]},
             return_train_score=True, scoring=make_scorer(rank_correl),
             verbose=1)
```

[27]:
```
joblib.dump(gridsearch_reg, results_path / 'rf_reg_gridsearch.joblib')
```

[27]:
```
['rf_reg_gridsearch.joblib']
```

[28]:
```
gridsearch_reg = joblib.load(results_path / 'rf_reg_gridsearch.joblib')
```

[29]:
```
gridsearch_reg.best_params_
```

[29]:
```
{'max_depth': 5, 'min_samples_leaf': 5, 'n_estimators': 50}
```

[30]:
```
f'{gridsearch_reg.best_score_*100:.2f}'
```

[30]:
```
'4.93'
```

1.3.5 Compare Results

Best Parameters

[31]:
```python
pd.DataFrame({'Regression': pd.Series(gridsearch_reg.best_params_),
              'Classification': pd.Series(gridsearch_clf.best_params_)})
```

[31]:

	Regression	Classification
max_depth	5	15
min_samples_leaf	5	5
n_estimators	50	100

Feature Importance

[32]:
```python
fi_clf = gridsearch_clf.best_estimator_.feature_importances_
fi_reg = gridsearch_reg.best_estimator_.feature_importances_
```

[33]:
```python
idx = [c.replace('_', ' ').upper() for c in X.columns]
```

[34]:
```python
fig, axes = plt.subplots(figsize=(14, 4), ncols=2)
(pd.Series(fi_clf, index=idx)
 .sort_values(ascending=False)
 .iloc[:15]
 .sort_values()
 .plot.barh(ax=axes[1], title='Classifier'))
(pd.Series(fi_reg, index=idx)
 .sort_values(ascending=False)
 .iloc[:15]
 .sort_values()
 .plot.barh(ax=axes[0], title='Regression'))
sns.despine()
fig.tight_layout()
```