02_sklearn_gbm_tuning

September 29, 2021

1 GBM Hyperparameter Tuning with sklearn

1.1 Imports & Settings

[2]:
```
from time import time
import numpy as np
import pandas as pd
import warnings
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from itertools import product
from sklearn.externals import joblib
from pathlib import Path

warnings.filterwarnings('ignore')
np.random.seed(42)
```

1.2 Create one-hot encoding

[5]:
```
def get_one_hot_data(df, cols=('year', 'month', 'age', 'msize')):
    cols = list(cols)
    df = pd.get_dummies(df,
                        columns=cols + ['sector'],
                        prefix=cols + [''],
                        prefix_sep=['_'] * len(cols) + [''])
    return df.rename(columns={c: c.replace('.0', '').replace(' ', '_').lower()␣
 ↪for c in df.columns})
```

1.3 Create holdout test set

[6]:
```
def get_holdout_set(target, features, period=6):
    idx = pd.IndexSlice
    label = target.name
    dates = np.sort(target.index.get_level_values('date').unique())
    cv_start, cv_end = dates[0], dates[-period - 2]
    holdout_start, holdout_end = dates[-period - 1], dates[-1]
```

```
    df = features.join(target.to_frame())
    train = df.loc[idx[:, cv_start: cv_end], :]
    y_train, X_train = train[label], train.drop(label, axis=1)

    test = df.loc[idx[:, holdout_start: holdout_end], :]
    y_test, X_test = test[label], test.drop(label, axis=1)
    return y_train, X_train, y_test, X_test
```

1.4 Custom TimeSeriesSplit

[8]:
```
class OneStepTimeSeriesSplit:
    """Generates tuples of train_idx, test_idx pairs
    Assumes the index contains a level labeled 'date'"""

    def __init__(self, n_splits=3, test_period_length=1, shuffle=False):
        self.n_splits = n_splits
        self.test_period_length = test_period_length
        self.shuffle = shuffle
        self.test_end = n_splits * test_period_length

    @staticmethod
    def chunks(l, n):
        for i in range(0, len(l), n):
            yield l[i:i + n]

    def split(self, X, y=None, groups=None):
        unique_dates = (X
                        .index
                        .get_level_values('date')
                        .unique()
                        .sort_values(ascending=False)
        [:self.test_end])

        dates = X.reset_index()[['date']]
        for test_date in self.chunks(unique_dates, self.test_period_length):
            train_idx = dates[dates.date < min(test_date)].index
            test_idx = dates[dates.date.isin(test_date)].index
            if self.shuffle:
                np.random.shuffle(list(train_idx))
            yield train_idx, test_idx

    def get_n_splits(self, X, y, groups=None):
        return self.n_splits
```

1.5 Instantiate GradientBoostingClassifier

[9]:
```
gb_clf = GradientBoostingClassifier(loss='deviance',
                                    learning_rate=0.1,
                                    n_estimators=100,
                                    subsample=1.0,
                                    criterion='friedman_mse',
                                    min_samples_split=2,
                                    min_samples_leaf=1,
                                    min_weight_fraction_leaf=0.0,
                                    max_depth=3,
                                    min_impurity_decrease=0.0,
                                    min_impurity_split=None,
                                    init=None,
                                    random_state=None,
                                    max_features=None,
                                    verbose=0,
                                    max_leaf_nodes=None,
                                    warm_start=False,
                                    presort='auto',
                                    validation_fraction=0.1,
                                    n_iter_no_change=None,
                                    tol=0.0001)
```

1.6 Load Data

We use the dataset generated by the notebook feature-engineering from Chapter 4 on Alpha Factor Research that needs to be executed first.

[3]:
```
DATA_STORE = Path('../../data/assets.h5')
```

[1]:
```
def get_data(start='2000', end='2018', holding_period=1, dropna=False):
    idx = pd.IndexSlice
    target = f'target_{holding_period}m'
    with pd.HDFStore(DATA_STORE) as store:
        df = store['engineered_features']

    if start is not None and end is not None:
        df = df.loc[idx[:, start: end], :]
    if dropna:
        df = df.dropna()

    y = (df[target] > 0).astype(int)
    X = df.drop([c for c in df.columns if c.startswith('target')], axis=1)
    return y, X
```

[ ]:
```
n_splits = 12
y, features = get_data()
X = get_one_hot_data(features).dropna()

y, X, y_test, X_test = get_holdout_set(target=y,
                                       features=X)
```

[ ]:
```
with pd.HDFStore('model_tuning.h5') as store:
    store.put('holdout/features', X_test)
    store.put('holdout/target', y_test)
```

1.7 Setup GridSearchCV

The GridSearchCV class in sklearn's model_selection module facilitates the systematic evaluation of all combinations of the hyperparameter values that we would like to test.

In the following code, we will illustrate this functionality for seven tuning parameters that will result in a total of 24 x 32 x 4 = 576 different model configurations.

1.7.1 Parameter Grid

First we define the cross-validation iterator:

cv = OneStepTimeSeriesSplit(n_splits=n_splits)

And next the parameter grid

[10]:
```
param_grid = dict(
        learning_rate=[.01, .1, .2],
        max_depth=list(range(3, 13, 3)),
        max_features=['sqrt', .8, 1],
        min_impurity_decrease=[0, .01],
        min_samples_split=[10, 50],
        n_estimators=[100, 300],
        subsample=[.8, 1],
)
```

[11]:
```
all_params = list(product(*param_grid.values()))
print('# Models = :', len(all_params))
```

```
# Models = : 576
```

1.7.2 Instantiate GridSearchCV

[ ]:
```
gs = GridSearchCV(gb_clf,
                  param_grid,
                  cv=cv,
                  scoring='roc_auc',
                  verbose=3,
```

```
                    n_jobs=-1,
                    return_train_score=True)
```

1.7.3 Fit GridSearchCV

This can take several days…

[ ]:
```
start = time()
gs.fit(X=X, y=y)
done = time()
```

1.7.4 Persist Results

[ ]:
```
print(f'Done in {done:.2f}s')
joblib.dump(gs, 'gbm_gridsearch.joblib')
```