第12章　組合預測器

第12章　組合預測器

- 組合演算法（Ensemble method）
- 投票組合器（Voting）
- 投票組合器（Voting）——軟投票法
- Bagging裝袋演算法
- 隨機森林演算法
- 強化組合器（Boosting）
- 梯度強化組合器，XGB組合器
- 各種組合器的網格搜尋

- 三個臭皮匠勝過一個諸葛亮，就是「組合演算法」的精神寫照，以下簡稱為「組合器」，它的想法很簡單：既然每個演算法都有其特長，能照顧資料的不同面向，如果將它們組合在一起，會不會得到一個更好的結果？

- 請問：如果要挑選演算法來組合時，要挑性質相近的？還是有點不相近的呢？答案是：不要太相近。因為太相近的話，那這三個人的意見還是等於一個人的意見。因此組合器的基本條件是：每個預測器之間要有一定程度的差異。

範例12-1 載入資料並完成預處理

程式碼

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.rcParams['font.sans-serif'] = ['DFKai-sb']
plt.rcParams['axes.unicode_minus'] = False
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings('ignore')
```

範例12-1 載入資料並完成預處理

承接上一頁

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
breast_cancer = load_breast_cancer()
df = pd.DataFrame(data = breast_cancer['data'],
                  columns = breast_cancer['feature_names'])
df['target'] = breast_cancer['target']
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size=0.33, random_state=2)
```

範例12-2 投票組合器——硬投票

程式碼

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
# 載入 VotingClassfier 投票組合器
from sklearn.ensemble import VotingClassifier
model_pl_lr = make_pipeline(StandardScaler(),
LogisticRegression())
model_pl_svc = make_pipeline(StandardScaler(), SVC())
model_pl_knn = make_pipeline(StandardScaler(),
KNeighborsClassifier())
```

範例12-2 投票組合器——硬投票

承接上一頁

```
model_pl_tree =
make_pipeline(DecisionTreeClassifier(max_depth=10))
vc = VotingClassifier([
    ('lr', model_pl_lr),
 ('svc', model_pl_svc),
    ('tree', model_pl_tree),
    ('knn', model_pl_knn)],
    voting='hard')
vc.fit(X_train, y_train)
train_score = vc.score(X_train, y_train)
test_score = vc.score(X_test, y_test)
print('訓練集的預測結果', train_score)
print('測試集的預測結果', test_score)
```

執行結果

```
訓練集的預測結果 0.992125984251968
測試集的預測結果 0.9734042553191489
```

範例12-3 投票組合器——軟投票

程式碼

```
model_pl_svc = make_pipeline(StandardScaler(),
SVC(probability=True))
vc = VotingClassifier([
    ('lr', model_pl_lr),
    ('svc', model_pl_svc),
    ('tree', model_pl_tree),
    ('knn', model_pl_knn)],
    voting='soft')
vc.fit(X_train, y_train)
train_score = vc.score(X_train, y_train)
test_score = vc.score(X_test, y_test)
print('訓練集的預測結果', train_score)
print('測試集的預測結果', test_score)
```

執行結果

```
訓練集的預測結果 0.994750656167979
測試集的預測結果 0.968085106382978 7
```

範例12-4 投票組合器——軟投票

程式碼

```
vc = VotingClassifier([
    ('lr', model_pl_lr),
    ('svc', model_pl_svc),
    ('tree', model_pl_tree),
    ('knn', model_pl_knn)],
    voting='soft', weights=[2, 2, 1, 2])
vc.fit(X_train, y_train)
train_score = vc.score(X_train, y_train)
test_score = vc.score(X_test, y_test)
print('訓練集的預測結果', train_score)
print('測試集的預測結果', test_score)
```

執行結果

```
訓練集的預測結果 0.994750656167979
測試集的預測結果 0.978723404255319
```

範例12-7 裝袋演算法

程式碼

```
from sklearn.ensemble import BaggingClassifier
bagc = BaggingClassifier(random_state=42, n_estimators=50)
bagc.fit(X_train, y_train)
print('訓練集的預測結果', bagc.score(X_train, y_train))
print('測試集的預測結果', bagc.score(X_test, y_test))
```

執行結果

```
訓練集的預測結果 1.0
測試集的預測結果 0.9574468085106383
```

範例12-8 隨機森林演算法

程式碼

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, n_estimators=50)
rfc.fit(X_train, y_train)
rfc.score(X_test, y_test)
print('訓練集的預測結果', rfc.score(X_train, y_train))
print('測試集的預測結果', rfc.score(X_test, y_test))
```

執行結果

```
訓練集的預測結果 1.0
測試集的預測結果 0.9521276595744681
```

範例12-9 用網格搜尋的方式探索隨機森林的最佳預測參數

程式碼

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'max_depth': [1,2,3,4],
    'n_estimators': [100, 300, 500]
}
rfc = RandomForestClassifier(random_state=42)
gs = GridSearchCV(rfc, param_grid=param_grid, cv=10)
gs.fit(X_train, y_train)
print('最佳參數', gs.best_params_)
print('訓練集的預測結果', gs.best_score_)
print('測試集的預測結果', gs.best_estimator_.score(X_test,
y_test))
```

執行結果

```
最佳參數 {'max_depth': 4, 'n_estimators': 300}
訓練集的預測結果 0.9607962213225372
測試集的預測結果 0.9521276595744681
```

範例12-10 強化組合器

程式碼

```
from sklearn.ensemble import AdaBoostClassifier
ada_clf = AdaBoostClassifier()
ada_clf.fit(X_train, y_train)
print('訓練集的預測結果', ada_clf.score(X_train, y_train))
print('測試集的預測結果', ada_clf.score(X_test, y_test))
```

執行結果

```
訓練集的預測結果 1.0
測試集的預測結果 0.9627659574468085
```

範例12-12 梯度強化組合器

程式碼

```
from sklearn.ensemble import GradientBoostingClassifier
gbc_clf = GradientBoostingClassifier(n_estimators=500)
gbc_clf.fit(X_train, y_train)
print('訓練集的預測結果', gbc_clf.score(X_train, y_train))
print('測試集的預測結果', gbc_clf.score(X_test, y_test))
```

執行結果

```
訓練集的預測結果 1.0
測試集的預測結果 0.9521276595744681
```

範例12-13 XGB 組合器

程式碼

```
from xgboost import XGBClassifier
xgb = XGBClassifier(n_estimators=500, max_depth=1,
learning_rate=0.05)
xgb.fit(X_train, y_train)
print('訓練集的預測結果', xgb.score(X_train, y_train))
print('測試集的預測結果', xgb.score(X_test, y_test))
```

執行結果

```
訓練集的預測結果 0.9973753280839895
測試集的預測結果 0.9627659574468085
```

範例12-14 各種組合器的網格搜尋

程式碼

```
from sklearn.pipeline import Pipeline
model_pl = Pipeline([
    ('preprocess', StandardScaler()),
    ('model', LogisticRegression())
])
param_grid = [
    {'model':[RandomForestClassifier()],
     'model_ _n_estimators': [100, 500]},
    {'model':[AdaBoostClassifier()],
     'model_ _n_estimators': [100, 500],
     'model_  _base_estimator':[None,
RandomForestClassifier(max_depth=1)]},
    {'model':[XGBClassifier()],
```

範例12-14 各種組合器的網格搜尋

承接上一頁

```
'model_ _n_estimators': [100, 500]},
]
gs = GridSearchCV(model_pl, param_grid=param_grid,
                  cv=5, return_train_score=True)
gs.fit(X_train, y_train)
score = gs.best_estimator_.score(X_test, y_test)
print('最佳模型', gs.best_params_['model'])
print('訓練集的最佳結果', gs.best_score_)
print('測試集的結果', score)
```

執行結果

```
最佳模型 AdaBoostClassifier(algorithm='SAMME.R', base_
                            estimator=None, learning_rate=1.0,
                    n_estimators=500, random_state=None)
訓練集的最佳結果 0.9737867395762132
測試集的結果 0.9680851063829787
```