Estimation, Detection and Analysis II

04 - Improvement Techniques

bagging

Random Forests

Boosting

bagging for rating

1. Create a *dataset* for random sorting

```
from sklearn.datasets import make_classification
X, y = make_classification ( n_samples =1000, n_features =20, n_informative =15,
    n_redundant =5, random_state =5)
```

2. set the model

```
from sklearn.ensemble import baggingclassifier
model = BaggageClassifier ()
```

3. Evaluate the model and show performance

```
from sklearn.model_selection import RepeatedStratifiedKFold
cv = RepeatedStratifiedKFold ( n_splits =10, n_repeats =3, random_state =1)

from sklearn.model_selection import cross_val_score
n_scores = cross_val_score ( model , X, y, scoring =' accuracy ', cv = cv , n_jobs =-1,
        error_score =' raise ')

from numpy import mean
from numpy import std
print(' Accuracy : %.3f (%.3f)' % ( mean ( n_scores ), std ( n_scores )))
```

Result:

```
Accuracy : 0.866 (0.043)
```

Note: Due to randomness, the result may vary

4. Fit the model *and* apply it to a new instance

```
model.fit ( X,y )
row = [[-4.7705504,-1.88685058,-0.96057964,2.53850317,-6.5843005,3.45711663,-
        7.46225013,2.01338213,-0.45086384,-1.89314931,-2.90675203,-0.21214568,-
        0.9623956,3.93862591,0.06276375,0.33964269,4.0835676,1.31423977,-
        2.17983117, 3.1047287]]
yhat = model.predict ( row )
print(' Predicted Class : %d' % yhat [0])
```

Result:

```
Predicted Grade : 1
```

Comparison of Bagging Methods in Classification

1. Import the iris *dataset* and split it into independent and dependent variables, training and testing

```
from sklearn import datasets
iris = datasets.load_iris ()

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split ( iris.data , iris.target )
```

2. Define the templates to use

```
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import perceptron
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
possible_base_estimators = [ DummyClassifier (),
                             Perceptron ( tol =1e-3),
                             DecisionTreeClassifier (),
                             KNeighborsClassifier ()]
```

3. Define the *grid* (*base model* , number of *base models*), run *bagging* and save the result

```
results = []
from sklearn.ensemble import baggingclassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from numpy import mean
for base_estimator in possible_base_estimators :
for n_estimators in [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]:
print( base_estimator , n_estimators )
        model = BaggageClassifier ( n_estimators = n_estimators ,
                 base_estimator = base_estimator , random_state =1)
        cv = RepeatedStratifiedKFold ( n_splits =10, n_repeats =3, random_state =1)
        n_scores = cross_val_score ( model , X_train , y_train , scoring =' accuracy ',
cv = cv ,
                 n_jobs =-1, error_score =' raise ')
        res = [ base_estimator , n_estimators , mean ( n_scores )]
        results.append ( res )
```

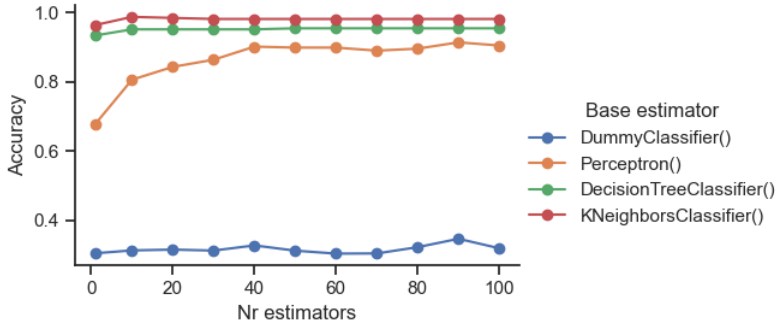
4. Convert the results into a *dataframe* and name the columns

```
from pandas import DataFrame
results = DataFrame ( results )
results.columns =['Base estimator ',' Nr estimators ', ' Accuracy ']
```

5. Graphically view the results

```
from  matplotlib import pyplot
import seaborn
seaborn.set ( style =' ticks ')
fg = seaborn.FacetGrid (data= results , hue ='Base estimator ', aspect =1.61)
fg.map ( pyplot.plot , ' Nr estimators ', ' Accuracy ', marker ='o'). add_legend ()
fg.savefig ('bagging_comp.png')
```

Result:



bagging for regression

1. Create a *dataset* for random regression

```
from sklearn.datasets import make_regression
X, y = make_regression ( n_samples =1000, n_features =20, n_informative =15, noise=0.1,
    random_state =5)
```

2. set the model

```
from sklearn.ensemble import BaggingRegressor
model = BaggingRegressor ()
```

3. Evaluate the model and show performance

```
from sklearn.model_selection import RepeatedKFold
cv = RepeatedKFold ( n_splits =10, n_repeats =3, random_state =1)

from sklearn.model_selection import cross_val_score
n_scores = cross_val_score ( model , X, y, scoring =' neg_mean_absolute_error ', cv = cv
,
       n_jobs =-1, error_score =' raise ')
from numpy import mean
from numpy import std
print('MAE: %.3f (%.3f)' % ( mean ( n_scores ), std ( n_scores )))
```

Result:

```
MAE: -101,255 (9,184)
```

Note: Due to randomness, the result may vary

4. Fit the model *and* apply it to a new instance

```
model.fit ( X,y )
row = [[0.88950817,-0.93540416,0.08392824,0.26438806,-0.52828711,-1.21102238,-
       0.4499934,1.47392391,-0.19737726,-0.22252503,0.02307668,0.26953276,0.03572757,-
       0.51606983,-0.39937452,1.8121736,-0.00775917,-0.02514283,-0.76089365,1.58692212]]
yhat = model.predict ( row )
print(' Prediction : %d' % yhat [0])
```

Result:

```
Prediction : -187
```

Random Forests

1. Import the iris *dataset* and split it into independent and dependent variables, training and testing

```
from sklearn import datasets
iris = datasets.load_iris ()

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split ( iris.data , iris.target )
```

2. Define the *grid* (number of *base models* and number of *features* to choose in each *split*), run *random forests* and save the result

```
results = []
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from numpy import mean
for n_estimator in [1, 50, 100, 250, 500, 750, 1000, 2500, 5000]:
for features in ['auto', ' sqrt ', 'log2']:
print( n_estimator , features )
        model = RandomForestClassifier ( n_estimators =100, max_features ="auto")
        cv = RepeatedStratifiedKFold ( n_splits =10, n_repeats =3, random_state =1)
        n_scores = cross_val_score ( model , X_train , y_train , scoring =' accuracy ',
cv = cv , n_jobs =-1, error_score =' raise ')
        res = [ n_estimator , features , mean ( n_scores )]
        results.append ( res )
```

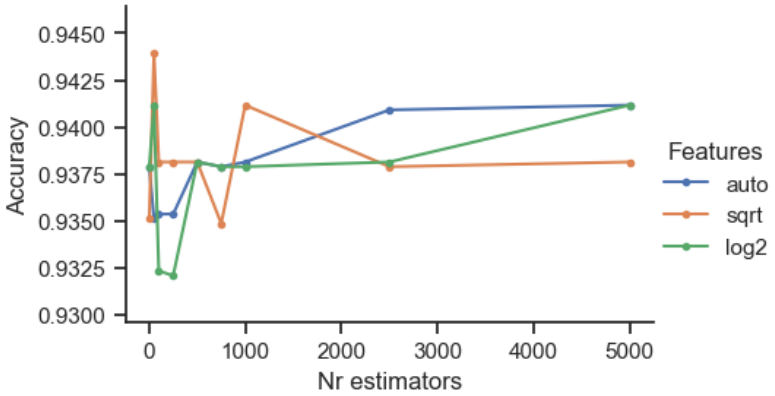
3. Convert the results into a *dataframe* and name the columns

```
from pandas import DataFrame
results = DataFrame ( results )
results.columns =[' Number estimators ', ' Features ', ' Accuracy ']
```

4. Graphically view the results

```
from matplotlib import pyplot
import seaborn
seaborn.set ( style =' ticks ')
fg = seaborn.FacetGrid (data= results , hue =' Features ', aspect =1.61)
fg.set ( ylim =( results [' Accuracy '].min()-0.0025, results [' Accuracy ']. max
()+0.0025))
fg.map ( pyplot.plot , ' Nr estimators ', ' Accuracy ', marker ='.'). add_legend ()
fg.savefig ('randomForests.png')
```

Result:



Note: Due to randomness, the result may vary

Boosting (*AdaBoost*)

1. Import the iris *dataset* and split it into independent and dependent variables, training and testing

```
from sklearn import datasets
iris = datasets.load_iris ()

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split ( iris.data , iris.target )
```

2. Define the *grid* (number of iterations), run *AdaBoost* (*default base learner* : *decision trees*) and save the result

```
results = []
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from numpy import mean
for n_estimator in [1, 10, 20, 30, 40, 50 , 60, 70, 80, 90, 100]:
print( n_estimator )
    model = AdaBoostClassifier ( n_estimators = n_estimator )
    cv = RepeatedStratifiedKFold ( n_splits =10, n_repeats =3, random_state =1)
    n_scores = cross_val_score ( model , X_train , y_train , scoring =' accuracy ', cv =
cv , n_jobs =-1, error_score =' raise ')
    res = [ n_estimator , mean ( n_scores )]
    results.append ( res )
```

3. Convert the results into a *dataframe* and name the columns

```
from pandas import DataFrame
results = DataFrame ( results )
results.columns =[' Number estimators ', ' Accuracy ']
```

4. Graphically view the results

```
from matplotlib import pyplot
import seaborn
seaborn.set ( style =' ticks ')
fg = seaborn.FacetGrid (data= results , aspect =1.61)
fg.set ( ylim =( results [' Accuracy '].min()-0.0025, results [' Accuracy ']. max
()+0.0025))
fg.map ( pyplot.plot , ' Nr estimators ', ' Accuracy ', marker ='.'). add_legend ()
fg.savefig ('adaboost.png')
```

Result: