Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Технологии машинного обучения»

Отчет по лабораторной работе №6:

«Ансамбли моделей машинного обучения»

Выполнил:
студент группы ИУ5-63
Курганова Александра

Подпись и дата:

Проверил:

Подпись и дата:

Москва, 2019 г.

Отчет по лаборторной работе №6 "Ансамбли моделей машинного обучения"

In [144]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, KFold, ShuffleSplit
from xgboost import XGBClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
%matplotlib inline
sns.set(style="ticks")
%matplotlib inline
sns.set(style='ticks')
```

подготовка датасета

In [145]:

```
data = pd.read_csv('Pokemon.csv')
```

In [146]:

```
data.head()
```

Out[146]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1

In [147]:

```
data.shape
```

Out[147]:

```
(800, 13)
```

In [148]:

```
data.isnull().sum()
```

Out[148]:

```
#               0
Name            0
Type 1          0
Type 2        386
Total           0
HP              0
Attack          0
Defense         0
Sp. Atk         0
Sp. Def         0
Speed           0
Generation      0
Legendary       0
dtype: int64
```

In [149]:

```
data.dtypes
```

Out[149]:

```
#              int64
Name          object
Type 1        object
Type 2        object
Total          int64
HP             int64
Attack         int64
Defense        int64
Sp. Atk        int64
Sp. Def        int64
Speed          int64
Generation     int64
Legendary       bool
dtype: object
```

In [150]:

```
data = data.drop(columns=['#', 'Name', 'Type 1', 'Type 2'])
```

In [151]:

```
data.isnull().sum()
```

Out[151]:

```
Total         0
HP            0
Attack        0
Defense       0
Sp. Atk       0
Sp. Def       0
Speed         0
Generation    0
Legendary     0
dtype: int64
```

In [152]:

```
x, y = data[data.columns[range(8)]], data[data.columns[[8]]]
print('x:', xc.columns)
print('y:', yc.columns)
```

```
x: Index(['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def',
'Speed',
       'Generation'],
      dtype='object')
y: Index(['Legendary'], dtype='object')
```

In [153]:

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3, random_stat
e=42)
len(xtrain), len(xtest), len(ytrain), len(ytest)
```

Out[153]:

```
(560, 240, 560, 240)
```

обучение моделей

BaggingClassifier with DecisionTreeClassifier

In [154]:

```
bc_trc = BaggingClassifier(DecisionTreeClassifier(random_state=42), n_estimators
=100)
bc_trc.fit(x, y.values.ravel())
```

Out[154]:

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight
=None,
                                                        criterion='g
ini',
                                                        max_depth=No
ne,
                                                        max_features
=None,
                                                        max_leaf_nod
es=None,
                                                        min_impurity
_decrease=0.0,
                                                        min_impurity
_split=None,
                                                        min_samples_
leaf=1,
                                                        min_samples_
split=2,
                                                        min_weight_f
raction_leaf=0.0,
                                                        presort=Fals
e,
                                                        random_state
=42,
                                                        splitter='be
st'),
                  bootstrap=True, bootstrap_features=False, max_feat
ures=1.0,
                  max_samples=1.0, n_estimators=100, n_jobs=None,
                  oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

In [155]:

```
bc_trctrain = bc_trc.predict(xtrain)
```

In [156]:

```
bc_trctest = bc_trc.predict(xtest)
```

In [157]:

```
print('train accuracy_score (%): {}'.format(accuracy_score(ytrain, bc_trctrain)))
print('test accuracy_score  ($): {}'.format(accuracy_score(ytest, bc_trctest)))
```

```
train accuracy_score (%): 0.9982142857142857
test accuracy_score  ($): 1.0
```

XGBClassifier

In [158]:

```
xg_trc = XGBClassifier(n_jobs=-1)
```

In [159]:

```
xg_trc.fit(x, y.values.ravel())
```

Out[159]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_
jobs=-1,
              nthread=None, objective='binary:logistic', random_stat
e=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=No
ne,
              silent=None, subsample=1, verbosity=1)
```

In [174]:

```
xg_trctrain = xg_trc.predict(xtrain)
```

In [175]:

```
xg_trctest = xg_trc.predict(xtest)
```

In [177]:

```
print('train accuracy_score (%): {}'.format(accuracy_score(ytrain, xg_trctrain
)))
print('test accuracy_score  (%): {}'.format(accuracy_score(ytest, xg_trctest)))
```

```
accuracy_score (train): 0.9964285714285714
accuracy_score  (test): 0.9958333333333333
```

подбор одного гиперпараметра с GridSearchCV и кросс-валидацией

In [163]:

```
param = [{'n_estimators': np.array(range(1, 101))}]
bctrc_grid = GridSearchCV(BaggingClassifier(DecisionTreeClassifier(random_state=
42)), param,
                          cv=KFold(n_splits=20), scoring='accuracy',
                          n_jobs=-1,
                         )
bctrc_grid.fit(x, y.values.ravel())
```

Out[163]:

```
GridSearchCV(cv=KFold(n_splits=20, random_state=None, shuffle=Fals
e),
             error_score='raise-deprecating',
             estimator=BaggingClassifier(base_estimator=DecisionTree
Classifier(class_weight=None,

criterion='gini',

max_depth=None,

max_features=None,

max_leaf_nodes=None,

min_impurity_decrease=0.0,

min_impurity_split=None,

min_samples_leaf=1,

min_samples_split=2,

min_weig...
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
39,
        40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
52,
        53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
65,
        66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
78,
        79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
91,
        92,  93,  94,  95,  96,  97,  98,  99, 100])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring='accuracy', verbose=0)
```

In [164]:

```
bctrc_grid.best_estimator_
```

Out[164]:

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight
=None,
                                                        criterion='g
ini',
                                                        max_depth=No
ne,
                                                        max_features
=None,
                                                        max_leaf_nod
es=None,
                                                        min_impurity
_decrease=0.0,
                                                        min_impurity
_split=None,
                                                        min_samples_
leaf=1,
                                                        min_samples_
split=2,
                                                        min_weight_f
raction_leaf=0.0,
                                                        presort=Fals
e,
                                                        random_state
=42,
                                                        splitter='be
st'),
                  bootstrap=True, bootstrap_features=False, max_feat
ures=1.0,
                  max_samples=1.0, n_estimators=84, n_jobs=None,
                  oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

In [165]:

```
bctrc_grid.best_score_
```

Out[165]:

```
0.95625
```

In [166]:

```
bctrc_grid.best_params_
```

Out[166]:

```
{'n_estimators': 84}
```

In [178]:

```
param = [{"colsample_bytree": [1.0], "min_child_weight": [0.8, 1.0, 1.2],
              'max_depth': range(3, 11), 'n_estimators': [25, 50, 75, 100]}]
xgbc_grid = GridSearchCV(XGBClassifier(), param, cv=KFold(n_splits=20), scoring=
'accuracy',
                           n_jobs=-1,
                          )
xgbc_grid.fit(x, y.values.ravel())
```

Out[178]:

```
GridSearchCV(cv=KFold(n_splits=20, random_state=None, shuffle=Fals
e),
             error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtre
e',
                                     colsample_bylevel=1, colsample_
bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_st
ep=0,
                                     max_depth=3, min_child_weight=
1,
                                     missing=None, n_estimators=100,
n_jobs=1,
                                     nthread=None, objective='...ist
ic',
                                     random_state=0, reg_alpha=0, re
g_lambda=1,
                                     scale_pos_weight=1, seed=None,
silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=-1,
             param_grid=[{'colsample_bytree': [1.0], 'max_depth': ra
nge(3, 11),
                          'min_child_weight': [0.8, 1.0, 1.2],
                          'n_estimators': [25, 50, 75, 100]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring='accuracy', verbose=0)
```

In [179]:

```
xgbc_grid.best_estimator_
```

Out[179]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1.0, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=7,
              min_child_weight=1.2, missing=None, n_estimators=25, n
_jobs=1,
              nthread=None, objective='binary:logistic', random_stat
e=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=No
ne,
              silent=None, subsample=1, verbosity=1)
```

In [180]:

```
xgbc_grid.best_score_
```

Out[180]:

```
0.95375
```

In [181]:

```
xgbc_grid.best_params_
```

Out[181]:

```
{'colsample_bytree': 1.0,
 'max_depth': 7,
 'min_child_weight': 1.2,
 'n_estimators': 25}
```

обучение снова с найденными гиперпараметрами

In [182]:

```
bctrc_grid.best_estimator_.fit(xtrain, ytrain.values.ravel())

bctrc_trainnew = bctrc_grid.best_estimator_.predict(xtrain)
bctrc_testnew = bctrc_grid.best_estimator_.predict(xtest)

print('train accuracy_score (%): {}'.format(accuracy_score(ytrain, bc_trctrain
)))
print('test accuracy_score (%): {}'.format(accuracy_score(ytest, bc_trctest)))

print('train accuracy_score (%): {}'.format(accuracy_score(ytrain, bctrc_trainne
w)))
print('test accuracy_score (%): {}'.format(accuracy_score(ytest, bctrc_testnew
)))
```

```
train accuracy_score (%): 0.9982142857142857
test accuracy_score (%): 1.0
train accuracy_score (%): 0.9982142857142857
test accuracy_score (%): 0.9666666666666667
```

In [183]:

```
xgbc_grid.best_estimator_.fit(xtrain, ytrain.values.ravel())

xgbc_trainnew = xgbc_grid.best_estimator_.predict(xtrain)
xgbc_testnew = xgbc_grid.best_estimator_.predict(xtest)

print('train accuracy_score (%): {}'.format(accuracy_score(ytrain, xg_trctrain
)))
print('test accuracy_score (%): {}'.format(accuracy_score(ytest, xg_trctest)))

print('train accuracy_score (%): {}'.format(accuracy_score(ytrain, xgbc_trainnew
)))
print('test accuracy_score (%): {}'.format(accuracy_score(ytest, xgbc_testnew)))
```

```
train accuracy_score (%): 0.9964285714285714
test accuracy_score (%): 0.9958333333333333
train accuracy_score (%): 0.9946428571428572
test accuracy_score (%): 0.9625
```