МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

Лабораторная работа 6

по дисциплине

«Методы машинного обучения»

по теме

«Ансамбли моделей машинного обучения»

ИСПОЛНИТЕЛЬ: Чечнев А.А.
ФИО

группа ИУ5-
23М

_____
подпись

"__"_____2020 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю. Е.
ФИО

_____
подпись

"__"_____2020 г.

Москва – 2020

Лабораторная работа

Ансамбли моделей машинного обучения.

Цель лабораторной работы: изучение ансамблей моделей машинного обучения.

Требования к отчету: Отчет по лабораторной работе должен содержать:

- титульный лист;
- описание задания;
- текст программы;
- экранные формы с примерами выполнения программы.

Задание:

- 1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
- 2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
- 3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
- 4. Обучите две ансамблевые модели. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.
- 5. Произведите для каждой модели подбор значений одного гиперпараметра. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
- 6. Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

In [1]:

```
#Start ML proj
import pandas as pd
pd.set_option('display.max.columns', 100)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(rc={'figure.figsize':(16,8)})
```

In [2]:

```
df = pd.read_csv('data/mlbootcamp5_train.csv', error_bad_lines=False, comment='#')
df.drop(columns='id', inplace = True);
```

In [3]:

```
df.head()
```

Out[3]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Преобразуем возраст в года

In [4]:

```
X = df.drop(columns='cardio')
X.age = X.age /255.25
y = df.cardio
```

Разобьем датасет на тренировочный и обучающий

In [5]:

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score, GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

Выбор метрик

Поскольку значения целевой переменной бинарны и равномерно распределены (50/50) то в качестве метрики воспользуемся точностью. Таже в качестве дополнительной метрики будем учитывать f1_score

In [11]:

```
from sklearn.metrics import f1_score as F1, accuracy_score as AS
```

Ансамблевые модели

Случайный лес

In [6]:

```
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier, RandomForestClassifier
```

In [15]:

```
clf_rf = RandomForestClassifier(n_estimators=100, max_depth=15)

%time clf_rf.fit(X_train, y_train)
```

```
Wall time: 4.41 s
```

Out[15]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=15, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [19]:

```
print('accuracy train:', AS(y_train, clf_rf.predict(X_train)))
print('accuracy test :', AS(y_test, clf_rf.predict(X_test)), '\n')

print('F1 train:', F1(y_train, clf_rf.predict(X_train)))
print('F1 test :', F1(y_test, clf_rf.predict(X_test)), '\n')
```

```
accuracy train: 0.8187959183673469
accuracy test : 0.7312857142857143

F1 train: 0.8084564771869269
F1 test : 0.7223616236162361
```

BaggingClassifier

In [23]:

```
clf_bc = BaggingClassifier(n_estimators=100)

%time clf_bc.fit(X_train, y_train)
```

```
Wall time: 19.2 s
```

Out[23]:

```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=Fa
lse,
                  max_features=1.0, max_samples=1.0, n_estimators=100,
                  n_jobs=None, oob_score=False, random_state=None, verbose=
0,
                  warm_start=False)
```

In [24]:

```
print('accuracy train:', AS(y_train, clf_bc.predict(X_train)))
print('accuracy test :', AS(y_test, clf_bc.predict(X_test)), '\n')

print('F1 train:', F1(y_train, clf_bc.predict(X_train)))
print('F1 test :', F1(y_test, clf_bc.predict(X_test)), '\n')
```

```
accuracy train: 0.9998163265306123
accuracy test : 0.711

F1 train: 0.999815350526251
F1 test : 0.7086273944980556
```

AdaBoostClassifier

In [29]:

```
clf_ab = AdaBoostClassifier(n_estimators=50)

%time clf_ab.fit(X_train, y_train)
```

```
Wall time: 1.45 s
```

Out[29]:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=
1.0,
                   n_estimators=50, random_state=None)
```

In [31]:

```
print('accuracy train:', AS(y_train, clf_ab.predict(X_train)))
print('accuracy test :', AS(y_test, clf_ab.predict(X_test)), '\n')

print('F1 train:', F1(y_train, clf_ab.predict(X_train)))
print('F1 test :', F1(y_test, clf_ab.predict(X_test)), '\n')
```

```
accuracy train: 0.7320612244897959
accuracy test : 0.7315238095238096

F1 train: 0.7096189149138522
F1 test : 0.714126356353311
```

GridSearch

In [37]:

```
from sklearn.model_selection import GridSearchCV

params = {'n_estimators' : np.arange(50, 150, 10),
          'max_depth' : np.arange(9, 10, 1)}

grcrch = GridSearchCV(estimator=RandomForestClassifier(), param_grid=params)
```

In [38]:

```
%time grcrch.fit(X_train, y_train)
```

```
Wall time: 4min 29s
```

Out[38]:

```
GridSearchCV(cv=None, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=No
ne,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': array([ 5,  7,  9, 11, 13, 15, 17, 1
9]),
                         'n_estimators': array([100, 101])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [42]:

```
grcrch.best_params_
```

Out[42]:

```
{'max_depth': 9, 'n_estimators': 101}
```

In [41]:

```
print('RandomForestClassifier \n')

print('accuracy train:', AS(y_train, grcrch.best_estimator_.predict(X_train)))
print('accuracy test :', AS(y_test, grcrch.best_estimator_.predict(X_test)), '\n')

print('F1 train:', F1(y_train, grcrch.best_estimator_.predict(X_train)))
print('F1 test :', F1(y_test, grcrch.best_estimator_.predict(X_test)), '\n')
```

```
RandomForestClassifier

accuracy train: 0.75
accuracy test : 0.7333809523809524

F1 train: 0.7334638816362055
F1 test : 0.7215951469345134
```

AdaBoostClassifier

In [43]:

```
params = {'n_estimators' : np.arange(10, 200, 30)}

grcrch = GridSearchCV(estimator=AdaBoostClassifier(), param_grid=params)
```

In [44]:

```
%time grcrch.fit(X_train, y_train)
```

```
Wall time: 1min 31s
```

Out[44]:

```
GridSearchCV(cv=None, error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
                                          base_estimator=None,
                                          learning_rate=1.0, n_estimators=5
0,
                                          random_state=None),
             iid='deprecated', n_jobs=None,
             param_grid={'n_estimators': array([ 10,  40,  70, 100, 130, 16
0, 190])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [45]:

```
print('RandomForestClassifier \n')

print('accuracy train:', AS(y_train, grcrch.best_estimator_.predict(X_train)))
print('accuracy test :', AS(y_test, grcrch.best_estimator_.predict(X_test)), '\n')

print('F1 train:', F1(y_train, grcrch.best_estimator_.predict(X_train)))
print('F1 test :', F1(y_test, grcrch.best_estimator_.predict(X_test)), '\n')
```

```
RandomForestClassifier

accuracy train: 0.7327959183673469
accuracy test : 0.7308095238095238

F1 train: 0.7095543379400608
F1 test : 0.7123008804519314
```

BaggingClassifier

In [46]:

```
params = {'n_estimators' : np.arange(10, 200, 40)}

grcrch = GridSearchCV(estimator=BaggingClassifier(), param_grid=params)
```

In [47]:

```
%time grcrch.fit(X_train, y_train)
```

```
Wall time: 6min 20s
```

Out[47]:

```
GridSearchCV(cv=None, error_score=nan,
             estimator=BaggingClassifier(base_estimator=None, bootstrap=Tru
e,
                                         bootstrap_features=False,
                                         max_features=1.0, max_samples=1.0,
                                         n_estimators=10, n_jobs=None,
                                         oob_score=False, random_state=None,
                                         verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'n_estimators': array([ 10,  50,  90, 130, 170])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [48]:

```
print('BaggingClassifier \n')

print('accuracy train:', AS(y_train, grcrch.best_estimator_.predict(X_train)))
print('accuracy test :', AS(y_test, grcrch.best_estimator_.predict(X_test)), '\n')

print('F1 train:', F1(y_train, grcrch.best_estimator_.predict(X_train)))
print('F1 test :', F1(y_test, grcrch.best_estimator_.predict(X_test)), '\n')
```

```
RandomForestClassifier

accuracy train: 0.9998367346938776
accuracy test : 0.7120952380952381

F1 train: 0.9998358705018259
F1 test : 0.7104960735491286
```

После подбора параметров улучшились показатели качества модели

Вывод: Таким образом мы рассмотрели и применили на практике ансамблевые модели