**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: |  | Проверил: |
| студент группы ИУ5-62 |  | преподаватель каф. ИУ5 |
| Король Константин |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2019 г.

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.

**Код программы**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score, recall\_score, f1\_score

from sklearn.metrics import make\_scorer

from sklearn.model\_selection import GridSearchCV

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

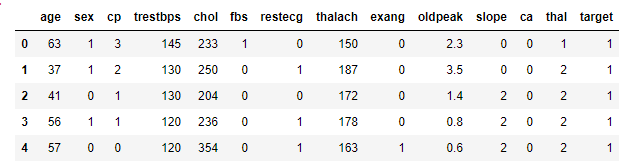
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import ExtraTreesClassifier

data = pd.read\_csv('data/heart.csv', sep=',')

data.head()



data.isnull().sum()

OUT: age 0

sex 0

cp 0

trestbps 0

chol 0

fbs 0

restecg 0

thalach 0

exang 0

oldpeak 0

slope 0

ca 0

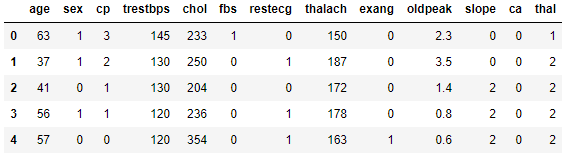
thal 0

target 0

dtype: int64

X = data.loc[:, data.columns != 'target']

X.head()



Y = data['target']

Y.head()

OUT: 0 1

1 1

2 1

3 1

4 1

Name: target, dtype: int64

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.2, random\_state = 1)

X\_train.shape, y\_train.shape

OUT: ((242, 13), (242,))

X\_test.shape, y\_test.shape

OUT: ((61, 13), (61,))

bagging\_tree = BaggingClassifier(DecisionTreeClassifier(random\_state=1), n\_estimators=100)

bagging\_tree.fit(X\_train, y\_train)

OUT: BaggingClassifier(base\_estimator=DecisionTreeClassifier(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\_weight\_fraction\_leaf=0.0, presort=False, random\_state=1,

splitter='best'),

bootstrap=True, bootstrap\_features=False, 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)

bagging\_tree\_y\_test = bagging\_tree.predict(X\_test)

bagging\_tree\_y\_test

OUT: array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,

1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,

0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0], dtype=int64)

classification\_report(y\_test, bagging\_tree\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, bagging\_tree\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.7307692307692307,

'recall': 0.6333333333333333,

'f1-score': 0.6785714285714285,

'support': 30},

{'precision': 0.6857142857142857,

'recall': 0.7741935483870968,

'f1-score': 0.7272727272727272,

'support': 31})

bagging\_tree\_n\_range = np.array(range(10, 200, 10))

bagging\_tree\_tuned\_parameters = [{'n\_estimators': bagging\_tree\_n\_range}]

bagging\_tree\_tuned\_parameters

OUT: [{'n\_estimators': array([ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130,

140, 150, 160, 170, 180, 190])}]

bagging\_tree\_gs = GridSearchCV(BaggingClassifier(DecisionTreeClassifier(random\_state=1)), bagging\_tree\_tuned\_parameters, cv=5, scoring=scoring, refit='f1')

bagging\_tree\_gs.fit(X\_train, y\_train)

OUT: GridSearchCV(cv=5, error\_score='raise-deprecating',

estimator=BaggingClassifier(base\_estimator=DecisionTreeClassifier(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,

...stimators=10, n\_jobs=None, oob\_score=False,

random\_state=None, verbose=0, warm\_start=False),

fit\_params=None, iid='warn', n\_jobs=None,

param\_grid=[{'n\_estimators': array([ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130,

140, 150, 160, 170, 180, 190])}],

pre\_dispatch='2\*n\_jobs', refit='f1', return\_train\_score='warn',

scoring={'recall': make\_scorer(recall\_score), 'f1': make\_scorer(f1\_score), 'accuracy': make\_scorer(accuracy\_score)},

verbose=0)

best\_bagging = bagging\_tree\_gs.best\_estimator\_

best\_bagging

OUT: BaggingClassifier(base\_estimator=DecisionTreeClassifier(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\_weight\_fraction\_leaf=0.0, presort=False, random\_state=1,

splitter='best'),

bootstrap=True, bootstrap\_features=False, max\_features=1.0,

max\_samples=1.0, n\_estimators=110, n\_jobs=None, oob\_score=False,

random\_state=None, verbose=0, warm\_start=False)

bagging\_tree\_gs.best\_score\_

OUT: 0.8778237616860984

bagging\_tree\_gs.best\_params\_

OUT: {'n\_estimators': 110}

best\_bagging.fit(X\_train, y\_train)

OUT: BaggingClassifier(base\_estimator=DecisionTreeClassifier(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\_weight\_fraction\_leaf=0.0, presort=False, random\_state=1,

splitter='best'),

bootstrap=True, bootstrap\_features=False, max\_features=1.0,

max\_samples=1.0, n\_estimators=110, n\_jobs=None, oob\_score=False,

random\_state=None, verbose=0, warm\_start=False)

best\_bagging\_y\_test = best\_bagging.predict(X\_test)

best\_bagging\_y\_test

OUT: array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,

1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,

0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0], dtype=int64)

In [40]:

classification\_report(y\_test, best\_bagging\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, best\_bagging\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.7692307692307693,

'recall': 0.6666666666666666,

'f1-score': 0.7142857142857142,

'support': 30},

{'precision': 0.7142857142857143,

'recall': 0.8064516129032258,

'f1-score': 0.7575757575757576,

'support': 31})

extra\_trees = ExtraTreesClassifier(random\_state=1, n\_estimators=100)

extra\_trees.fit(X\_train, y\_train)

OUT: ExtraTreesClassifier(bootstrap=False, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=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=1, verbose=0, warm\_start=False)

extra\_trees\_y\_test = extra\_trees.predict(X\_test)

extra\_trees\_y\_test

OUT: array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,

0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,

0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0], dtype=int64)

classification\_report(y\_test, extra\_trees\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, extra\_trees\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.7692307692307693,

'recall': 0.6666666666666666,

'f1-score': 0.7142857142857142,

'support': 30},

{'precision': 0.7142857142857143,

'recall': 0.8064516129032258,

'f1-score': 0.7575757575757576,

'support': 31})

extra\_tree\_n\_range = np.array(range(10, 200, 10))

extra\_tree\_tuned\_parameters = [{'n\_estimators': extra\_tree\_n\_range}]

extra\_tree\_tuned\_parameters

OUT: [{'n\_estimators': array([ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130,

140, 150, 160, 170, 180, 190])}]

extra\_tree\_gs = GridSearchCV(ExtraTreesClassifier(random\_state=1), extra\_tree\_tuned\_parameters, cv=5, scoring=scoring, refit='f1')

extra\_tree\_gs.fit(X\_train, y\_train)

OUT: GridSearchCV(cv=5, error\_score='raise-deprecating',

estimator=ExtraTreesClassifier(bootstrap=False, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=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='warn', n\_jobs=None,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False),

fit\_params=None, iid='warn', n\_jobs=None,

param\_grid=[{'n\_estimators': array([ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130,

140, 150, 160, 170, 180, 190])}],

pre\_dispatch='2\*n\_jobs', refit='f1', return\_train\_score='warn',

scoring={'recall': make\_scorer(recall\_score), 'f1': make\_scorer(f1\_score), 'accuracy': make\_scorer(accuracy\_score)},

verbose=0)

best\_extra\_tree = extra\_tree\_gs.best\_estimator\_

best\_extra\_tree

OUT: ExtraTreesClassifier(bootstrap=False, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=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=140, n\_jobs=None,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False)

extra\_tree\_gs.best\_params\_

OUT: {'n\_estimators': 140}

extra\_tree\_gs.best\_score\_

OUT: 0.8781143731089153

best\_extra\_tree.fit(X\_train, y\_train)

OUT: ExtraTreesClassifier(bootstrap=False, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=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=140, n\_jobs=None,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False)

best\_extra\_tree\_y\_test = best\_extra\_tree.predict(X\_test)

best\_extra\_tree\_y\_test

OUT: array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,

1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0,

0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0], dtype=int64)

classification\_report(y\_test, best\_extra\_tree\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, best\_extra\_tree\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.7916666666666666,

'recall': 0.6333333333333333,

'f1-score': 0.7037037037037038,

'support': 30},

{'precision': 0.7027027027027027,

'recall': 0.8387096774193549,

'f1-score': 0.7647058823529411,

'support': 31})

# для сравнения лучший результат беггинга

classification\_report(y\_test, best\_bagging\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, best\_bagging\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.7692307692307693,

'recall': 0.6666666666666666,

'f1-score': 0.7142857142857142,

'support': 30},

{'precision': 0.7142857142857143,

'recall': 0.8064516129032258,

'f1-score': 0.7575757575757576,

'support': 31})

# таким образом, с небольшим перевесом себя лучше показал беггинг

# однако, лучший результат среди всех методов из 5 и 6 ЛР показал метод логистической регрессии с параметрами по умолчанию

classification\_report(y\_test, logistic\_y\_test, output\_dict=True)["0"], \

classification\_report(y\_test, logistic\_y\_test, output\_dict=True)["1"]

OUT: ({'precision': 0.8,

'recall': 0.6666666666666666,

'f1-score': 0.7272727272727272,

'support': 30},

{'precision': 0.7222222222222222,

'recall': 0.8387096774193549,

'f1-score': 0.7761194029850746,

'support': 31})