Рубежный контроль №2

Методы построения моделей машинного обучения

Алиев Тимур РТ5-61Б вариант №1

Ввод [1]:

```
# Импорт библиотек
from operator import itemgetter
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.tree import DecisionTreeRegressor, export_graphviz, export_text
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import AdaBoostClassifier

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import graphviz

%matplotlib inline
sns.set(style='ticks')

```

Ввод [2]:

```
# Загрузка датасета
boston = load_boston()
```

```
C:\Users\Truma\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Functi
on load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Ввод [3]:
```
# Наименования признаков
boston.feature_names
```

Out[3]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

Ввод [4]:
```
# Значения признаков
boston.data[:5]
```

Out[4]:
```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, 0.0000e+00, 5.3800e-01,
        6.5750e+00, 6.5200e+01, 4.0900e+00, 1.0000e+00, 2.9600e+02,
        1.5300e+01, 3.9690e+02, 4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
        6.4210e+00, 7.8900e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
        1.7800e+01, 3.9690e+02, 9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
        7.1850e+00, 6.1100e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
        1.7800e+01, 3.9283e+02, 4.0300e+00],
       [3.2370e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 4.5800e-01,
        6.9980e+00, 4.5800e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
        1.8700e+01, 3.9463e+02, 2.9400e+00],
       [6.9050e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 4.5800e-01,
        7.1470e+00, 5.4200e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
        1.8700e+01, 3.9690e+02, 5.3300e+00]])
```

Ввод [5]:
```
# Значение целевого признака
np.unique(boston.target)
```

Out[5]:
```
array([ 5. ,  5.6,  6.3,  7. ,  7.2,  7.4,  7.5,  8.1,  8.3,  8.4,  8.5,
        8.7,  8.8,  9.5,  9.6,  9.7, 10.2, 10.4, 10.5, 10.8, 10.9, 11. ,
       11.3, 11.5, 11.7, 11.8, 11.9, 12. , 12.1, 12.3, 12.5, 12.6, 12.7,
       12.8, 13. , 13.1, 13.2, 13.3, 13.4, 13.5, 13.6, 13.8, 13.9, 14. ,
       14.1, 14.2, 14.3, 14.4, 14.5, 14.6, 14.8, 14.9, 15. , 15.1, 15.2,
       15.3, 15.4, 15.6, 15.7, 16. , 16.1, 16.2, 16.3, 16.4, 16.5, 16.6,
       16.7, 16.8, 17. , 17.1, 17.2, 17.3, 17.4, 17.5, 17.6, 17.7, 17.8,
       17.9, 18. , 18.1, 18.2, 18.3, 18.4, 18.5, 18.6, 18.7, 18.8, 18.9,
       19. , 19.1, 19.2, 19.3, 19.4, 19.5, 19.6, 19.7, 19.8, 19.9, 20. ,
       20.1, 20.2, 20.3, 20.4, 20.5, 20.6, 20.7, 20.8, 20.9, 21. , 21.1,
       21.2, 21.4, 21.5, 21.6, 21.7, 21.8, 21.9, 22. , 22.1, 22.2, 22.3,
       22.4, 22.5, 22.6, 22.7, 22.8, 22.9, 23. , 23.1, 23.2, 23.3, 23.4,
       23.5, 23.6, 23.7, 23.8, 23.9, 24. , 24.1, 24.2, 24.3, 24.4, 24.5,
       24.6, 24.7, 24.8, 25. , 25.1, 25.2, 25.3, 26.2, 26.4, 26.5, 26.6,
       26.7, 27. , 27.1, 27.5, 27.9, 28. , 28.1, 28.2, 28.4, 28.5, 28.6,
       28.7, 29. , 29.1, 29.4, 29.6, 29.8, 29.9, 30.1, 30.3, 30.5, 30.7,
       30.8, 31. , 31.1, 31.2, 31.5, 31.6, 31.7, 32. , 32.2, 32.4, 32.5,
       32.7, 32.9, 33. , 33.1, 33.2, 33.3, 33.4, 33.8, 34.6, 34.7, 34.9,
       35.1, 35.2, 35.4, 36. , 36.1, 36.2, 36.4, 36.5, 37. , 37.2, 37.3,
       37.6, 37.9, 38.7, 39.8, 41.3, 41.7, 42.3, 42.8, 43.1, 43.5, 43.8,
       44. , 44.8, 45.4, 46. , 46.7, 48.3, 48.5, 48.8, 50. ])
```

Ввод [6]:
```
# Размер выборки
boston.data.shape, boston.target.shape
```

Out[6]:
```
((506, 13), (506,))
```

Ввод [7]:
```
# Сформируем DataFrame
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

raw_df.rename(columns={0: 'CRIM'}, inplace=True)
raw_df.rename(columns={1: 'ZN'}, inplace=True)
raw_df.rename(columns={2: 'INDUS'}, inplace=True)
raw_df.rename(columns={3: 'CHAS'}, inplace=True)
raw_df.rename(columns={4: 'NOX'}, inplace=True)
raw_df.rename(columns={5: 'RM'}, inplace=True)
raw_df.rename(columns={6: 'AGE'}, inplace=True)
raw_df.rename(columns={7: 'DIS'}, inplace=True)
raw_df.rename(columns={8: 'RAD'}, inplace=True)
raw_df.rename(columns={9: 'TAX'}, inplace=True)
raw_df.rename(columns={10: 'PTRATIO'}, inplace=True)
```

Ввод [8]:
```
# Удаление строк, содержащих пустые значения
raw_df_2 = raw_df.dropna(axis=0, how='any')
(raw_df.shape, raw_df_2.shape)
```

Out[8]:
```
((1012, 11), (506, 11))
```

Ввод [9]:
```
# Проверим наличие пустых значений
# Цикл по колонкам датасета
for col in raw_df_2.columns:
    # Кол-во путсых значений - все значения заполнены
    temp_null_count = raw_df_2[raw_df_2[col].isnull()].shape[0]
    print ('{} - {}'.format(col, temp_null_count))
```

```
CRIM - 0
ZN - 0
INDUS - 0
CHAS - 0
NOX - 0
RM - 0
AGE - 0
DIS - 0
RAD - 0
TAX - 0
PTRATIO - 0
```

Ввод [10]:

```
raw_df_2['target'] = target
target1 = np.empty(len(raw_df_2['target']), dtype=np.int16)
j = 0
a = [0, 1, 2]
for i in raw_df_2['target']:
    if i <= 20 and i >= 5:
        target1[j] = 0
        j = j + 1
    if i <= 35 and i > 20:
        target1[j] = 1
        j = j + 1
    if i <= 50 and i > 35:
        target1[j] = 2
        j = j + 1
raw_df_2['target1'] = target1
raw_df_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 506 entries, 0 to 1010
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  target   506 non-null    float64
 12  target1  506 non-null    int16
dtypes: float64(12), int16(1)
memory usage: 52.4 KB
```

```
C:\Users\Truma\AppData\Local\Temp\ipykernel_5036\200341314.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  raw_df_2['target'] = target
C:\Users\Truma\AppData\Local\Temp\ipykernel_5036\200341314.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  raw_df_2['target1'] = target1
```

Ввод [11]:

```
boston_X_train, boston_X_test, boston_y_train, boston_y_test = train_test_split(
    raw_df_2.drop(['target1'], axis=1), raw_df_2['target1'], test_size=0.5, random_state=17)
```

Ввод [12]:

```
def print_metrics(y_true, y_pred):
    """
    Функия для оценки качества модели

    :param y_true: Истинные значения целевого признака
    :param y_pred: Предсказанные значения целевого признака
    """

    print('Accuracy: {};'.format(accuracy_score(y_true, y_pred)))
    print('Precision: {};'.format(precision_score(y_true, y_pred, average='weighted')))
    print('Recall: {};'.format(recall_score(y_true, y_pred, average='weighted')))
    print('F1-score: {}.'.format(f1_score(y_true, y_pred, average='weighted')))
```

Ввод [13]:

```
scaler = StandardScaler().fit(boston_X_train)
boston_X_train_scaled = pd.DataFrame(scaler.transform(boston_X_train), columns=boston_X_train.columns)
boston_X_test_scaled = pd.DataFrame(scaler.transform(boston_X_test), columns=boston_X_train.columns)
boston_X_train_scaled.describe()
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	P
count	2.530000e+02	2.530000e+02	2.530000e+02	253.000000	2.530000e+02	2.530000e+02	2.530000e+02	2.530000e+02	2.530000e+02	2.530000e+02	2.530
mean	5.616939e-17	-7.021173e-18	-5.967997e-17	0.000000	2.281881e-16	-1.011049e-15	-5.757362e-16	-2.106352e-16	-3.510587e-17	1.843058e-17	4.844
std	1.001982e+00	1.001982e+00	1.001982e+00	1.001982	1.001982e+00	1.001982e+00	1.001982e+00	1.001982e+00	1.001982e+00	1.001982e+00	1.001
min	-4.896445e-01	-4.765149e-01	-1.585390e+00	-0.276759	-1.411455e+00	-3.742422e+00	-2.362153e+00	-1.258854e+00	-1.001348e+00	-1.284044e+00	-2.667
25%	-4.778570e-01	-4.765149e-01	-8.681584e-01	-0.276759	-9.555808e-01	-5.981299e-01	-7.768940e-01	-8.206675e-01	-6.646074e-01	-7.683696e-01	-4.805
50%	-4.474570e-01	-4.765149e-01	-2.232402e-01	-0.276759	-1.600349e-01	-1.456454e-01	2.894877e-01	-2.232697e-01	-5.523606e-01	-4.670769e-01	2.029
75%	1.132433e-01	1.022018e-01	1.017895e+00	-0.276759	6.712659e-01	5.432726e-01	9.014046e-01	6.066735e-01	1.580328e+00	1.485531e+00	7.953
max	8.782363e+00	4.153219e+00	2.440552e+00	3.613247	2.816558e+00	3.245951e+00	1.112534e+00	3.370167e+00	1.580328e+00	1.746265e+00	1.615

Дерево решений

Ввод [14]:

```
# Гиперпараметры для оптимизации
parameters_to_tune = {'max_depth' : np.arange(1, 6, 1), # 1-5
                      'min_samples_leaf' : np.linspace(0.02, 0.1, 5),
                      'max_features' : [0.2 , 0.4, 0.6, 0.8, 'auto', 'sqrt', 'log2']}
```

Ввод [15]:

```
%%time
# Оптимизация гиперпараметров
dtc_gs = GridSearchCV(DecisionTreeClassifier(random_state=3),
                      parameters_to_tune, cv=5, scoring='accuracy')
dtc_gs.fit(boston_X_train, boston_y_train)
```

```
CPU times: total: 2.2 s
Wall time: 2.19 s
```

Out[15]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=3),
             param_grid={'max_depth': array([1, 2, 3, 4, 5]),
                         'max_features': [0.2, 0.4, 0.6, 0.8, 'auto', 'sqrt',
                                          'log2'],
                         'min_samples_leaf': array([0.02, 0.04, 0.06, 0.08, 0.1 ])},
             scoring='accuracy')
```

Ввод [16]:
```
# Лучшее значение параметров
dtc_gs.best_params_
```

Out[16]:
```
{'max_depth': 2, 'max_features': 0.6, 'min_samples_leaf': 0.02}
```

Ввод [17]:
```
# Лучшее значение метрики
dtc_gs.best_score_
```

Out[17]:
```
1.0
```

Ввод [18]:
```
# Обучение модели
dt_classifier : DecisionTreeClassifier = dtc_gs.best_estimator_
dt_classifier.fit(boston_X_train, boston_y_train)
```

Out[18]:
```
DecisionTreeClassifier(max_depth=2, max_features=0.6, min_samples_leaf=0.02,
                       random_state=3)
```

Ввод [19]:
```
# Предсказания модели регрессора на основе дерева решений
dt_pred = dt_classifier.predict(boston_X_test)
```

Градиентный бустинг

Ввод [20]:
```
# Гиперпараметры для оптимизации
parameters_to_tune = {'n_estimators' : [2, 5, 10],
                      'learning_rate': np.linspace(0.1, 0.3, 3),
                      'min_samples_split': np.arange(2, 5, 1), # 2-4
                      'max_depth' : np.arange(1, 5, 1)} # 1-4
```

Ввод [21]:
```
%%time
# Оптимизация гиперпараметров
gbc_gs = GridSearchCV(GradientBoostingClassifier(random_state=3),
                      parameters_to_tune, cv=5, scoring='accuracy')
gbc_gs.fit(boston_X_train, boston_y_train)
```

```
CPU times: total: 6.12 s
Wall time: 6.2 s
```

Out[21]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(random_state=3),
             param_grid={'learning_rate': array([0.1, 0.2, 0.3]),
                         'max_depth': array([1, 2, 3, 4]),
                         'min_samples_split': array([2, 3, 4]),
                         'n_estimators': [2, 5, 10]},
             scoring='accuracy')
```

Ввод [22]:
```
# Лучшее значение параметров
gbc_gs.best_params_
```

Out[22]:
```
{'learning_rate': 0.1,
 'max_depth': 1,
 'min_samples_split': 2,
 'n_estimators': 10}
```

Ввод [23]:
```
# Лучшее значение метрики
gbc_gs.best_score_
```

Out[23]:
```
1.0
```

Ввод [24]:
```
# Обучение модели
gb_classifier : GradientBoostingClassifier = gbc_gs.best_estimator_
gb_classifier.fit(boston_X_train, boston_y_train)
```

Out[24]:
```
GradientBoostingClassifier(max_depth=1, n_estimators=10, random_state=3)
```

Ввод [25]:
```
# Предсказания модели регрессора на основе градиентного бустинга
gb_pred = gb_classifier.predict(boston_X_test)
```

Оценка качества моделей

Ввод [26]:
```
# Оценка качества дерева решений
print('Дерево решений')
print_metrics(boston_y_test, dt_pred)
```

```
Дерево решений
Accuracy: 1.0;
Precision: 1.0;
Recall: 1.0;
F1-score: 1.0.
```

Ввод [27]:
```
# Оценка качества градиентного бустинга
print('Градиентный бустинг')
print_metrics(boston_y_test, gb_pred)
```

```
Градиентный бустинг
Accuracy: 1.0;
Precision: 1.0;
Recall: 1.0;
F1-score: 1.0.
```

Построенные модели с одинаково высокой точностью решают задачу классификации.

Градиентный бустинг невосприимчив к переобучению.