Демьянчук Г.В. ИУ5-22М Л/Р №6

Цель лабораторной работы

Изучить ансамбли моделей машинного обучения.

Задание

Требуется выполнить следующие действия:

1. Выбрать набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование к
3. С использованием метода `train_test_split` разделите выборку на обучающую и тестову
4. Обучите две ансамблевые модели. Оцените качество модели с помощью одной из подхо качество полученных моделей.
5. Произведите для каждой модели подбор одного гиперпараметра. В зависимости от испо применять функцию `GridSearchCV`, использовать перебор параметров в цикле, или испол
6. Повторите пункт 4 для найденных оптимальных значения гиперпараметров. Сравните ка с качеством моделей, полученных в пункте 4.

Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков:

```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Enable inline plots
%matplotlib inline
```

```
# Set plots formats to save high resolution PNG
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

Зададим ширину текстового представления данных, чтобы в дальнейшем текст в отчёте влеза

```
pd.set_option("display.width", 70)
```

Предварительная подготовка данных

В качестве набора данных используется оценка качества белых вин по шкале с учетом химич

```
data = pd.read_csv("/content/whitew.csv")
```

Проверим полученные типы:

```
data.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

Посмотрим на данные в данном наборе данных:

```
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free su dio
0	7.4	0.70	0.00	1.9	0.076	

```
df = data.copy()
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free su dio
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

```
df.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

С такими данными уже можно работать. Проверим размер набора данных:

```
df.shape
```

```
(1599, 12)
```

Проверим основные статистические характеристики набора данных:

```
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	s di
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0

Проверим наличие пропусков в данных:

```
df.isnull().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Разделение данных

Разделим данные на целевой столбец и признаки:

```
X = df.drop("density", axis=1)
y = df["density"]


print(X.head(), "\n")
print(y.head())
```

```
   fixed acidity  volatile acidity  citric acid  ...  sulphates  alcohol  quality
0            7.4              0.70         0.00  ...       0.56      9.4        5
1            7.8              0.88         0.00  ...       0.68      9.8        5
2            7.8              0.76         0.04  ...       0.65      9.8        5
3           11.2              0.28         0.56  ...       0.58      9.8        6
4            7.4              0.70         0.00  ...       0.56      9.4        5

[5 rows x 11 columns]

0    0.9978
1    0.9968
```

```
print(X.shape)
print(y.shape)
```

```
(1599, 11)
(1599,)
```

Предобработаем данные, чтобы методы работали лучше:

```
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sul diox
count	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e
mean	3.435512e-16	1.699704e-16	4.335355e-16	-1.905223e-16	4.838739e-16	1.432042e
std	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e
min	-2.137045e+00	-2.278280e+00	-1.391472e+00	-1.162696e+00	-1.603945e+00	-1.422500e
25%	-7.007187e-01	-7.699311e-01	-9.293181e-01	-4.532184e-01	-3.712290e-01	-8.487156e
50%	-2.410944e-01	-4.368911e-02	-5.636026e-02	-2.403750e-01	-1.799455e-01	-1.793002e
75%	5.057952e-01	6.266881e-01	7.652471e-01	4.341614e-02	5.384542e-02	4.901152e
max	4.355149e+00	5.877976e+00	3.743574e+00	9.195681e+00	1.112703e+01	5.367284e

Разделим выборку на тренировочную и тестовую:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25, random_state=346705925)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
print(y_train.shape)
print(y_test.shape)
```

```
(1199, 11)
(400, 11)
(1199,)
(400,)
```

Обучение моделей

Напишем функцию, которая считает метрики построенной модели:

```
def test_model(model):
    print("mean_absolute_error:",
          mean_absolute_error(y_test, model.predict(X_test)))
    print("median_absolute_error:",
          median_absolute_error(y_test, model.predict(X_test)))
    print("r2_score:",
          r2_score(y_test, model.predict(X_test)))
```

Случайный лес

Попробуем случайный лес с гиперпараметром $n = 100$:

```
ran_100 = RandomForestRegressor(n_estimators=100)
ran_100.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, 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)
```

Проверим метрики построенной модели:

```
test_model(ran_100)
```

```
mean_absolute_error: 0.0004663771922352339
median_absolute_error: 0.00032255390892999003
r2_score: 0.8825655447277275
```

Видно, что данный метод даже без настройки гиперпараметров уже показывает очень неплох

Градиентный бустинг

Попробуем градиентный бустинг с гиперпараметром $n = 100$:

```
gr_100 = GradientBoostingRegressor(n_estimators=100)
gr_100.fit(X_train, y_train)
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          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, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

Проверим метрики построенной модели:

```
test_model(gr_100)
```

```
mean_absolute_error: 0.0004704435493496417
median_absolute_error: 0.00034447129988390834
r2_score: 0.8910730667571881
```

Внезапно градиентный бустинг оказался несколько лучше по сравнению со случайным лесом

Подбор гиперпараметра n

Случайный лес

Введем список настраиваемых параметров:

```
param_range = np.arange(10, 201, 10)
tuned_parameters = [{'n_estimators': param_range}]
tuned_parameters
```

```
[{'n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130
              140, 150, 160, 170, 180, 190, 200])}]
```

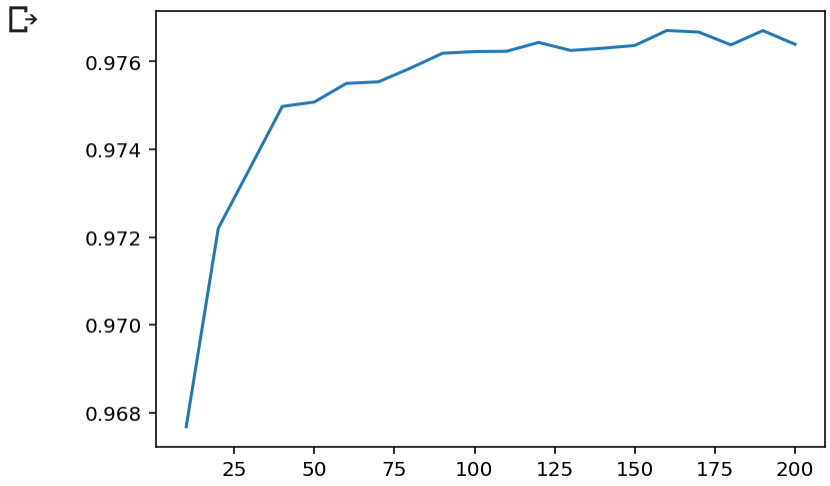
Запустим подбор параметра:

```
gs = GridSearchCV(RandomForestRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, 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=190, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данны

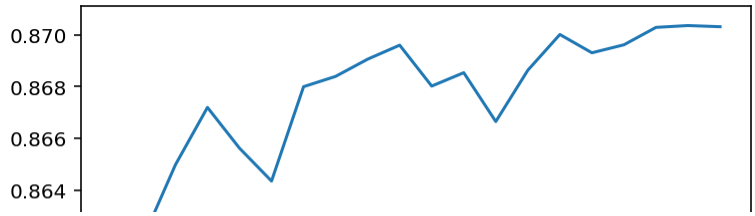
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



В целом результат ожидаемый — чем больше обученных моделей, тем лучше.

На тестовом наборе данных картина похожа:

```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Из-за случайнойсти график немного плавает, но конкретно в данном случае получился чётко и результатом.

```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.00046906389307640653
median_absolute_error: 0.00033821622168678234
r2_score: 0.8833787278965249
```

Конкретно данная модель оказалась практически такой же, как исходная.

Градиентный бустинг

Список настраиваемых параметров оставим тем же.

```
tuned_parameters
```

```
[{'n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130
        140, 150, 160, 170, 180, 190, 200])}]
```

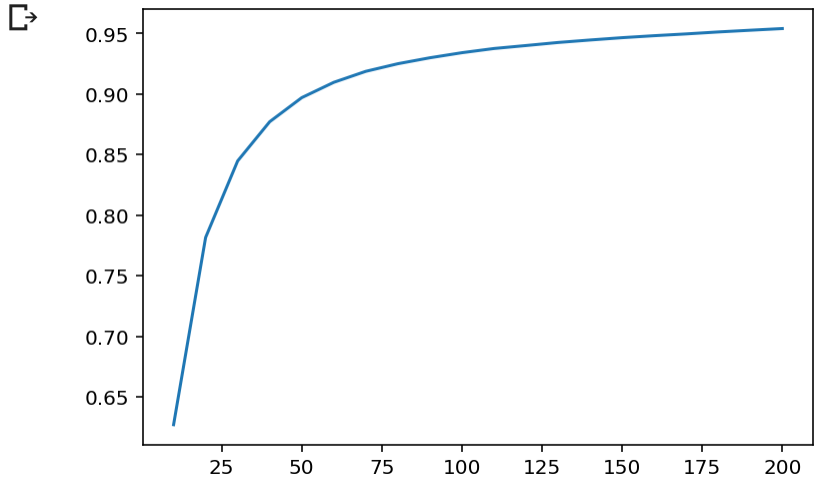
Запустим подбор параметра:

```
gs = GridSearchCV(GradientBoostingRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          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, n_estimators=200,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данны

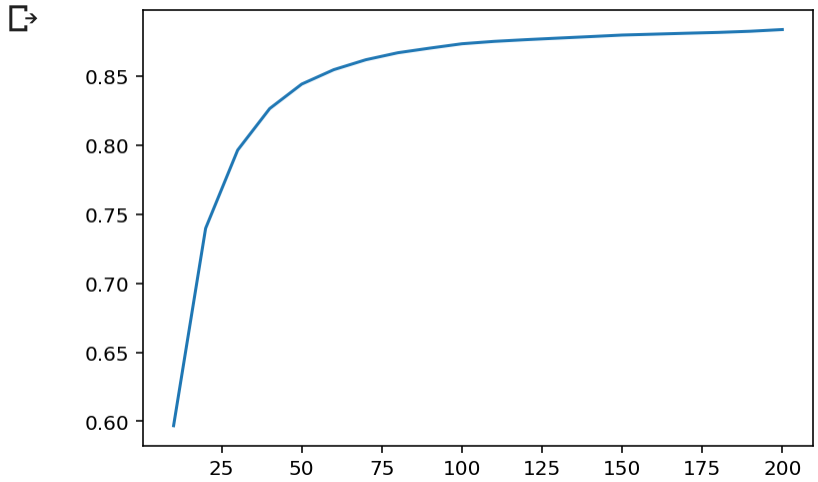
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



Картина та же: чем больше подмоделей, тем лучше.

На тестовом наборе данных картина ровно та же:

```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Выходит, что чем больше подмоделей, тем лучше. Возможно, что можно использовать ещё бо
это выходит за рамки лабораторной работы.

```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.0004392660468100229
median_absolute_error: 0.00032974298995036566
r2_score: 0.9053861296404554
```