

Лабораторная работа №6
по дисциплине
«Методы машинного обучения»
на тему
«Ансамбли моделей машинного обучения»

Выполнил:
студент группы ИУ5-22М
Бурашников В. В.

1. Цель лабораторной работы

Изучить ансамбли моделей машинного обучения [?].

2. Задание

Требуется выполнить следующие действия [?]:

1. Выбрать набор данных (датасет) для решения задачи классификации или регрессии.
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 модели с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.
5. Произведите для каждой модели подбор 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 значения гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

3. Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков [?, ?]:

```
[3]: 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 влезал на A4 [?]:

```
[4]: pd.set_option("display.width", 70)
```

3.1. Предварительная подготовка данных

В качестве набора данных используются метрологические данные с метеостанции HI-SEAS (Hawaii Space Exploration Analog and Simulation) за четыре месяца (с сентября по декабрь 2016 года) [?]:

```
[5]: data = pd.read_csv("./wine.csv")
```

Проверим полученные типы:

```
[6]: data.dtypes
```

```
[6]: 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:

```
[7]: data.head()
```

```
[7]:   fixed acidity  volatile acidity  citric acid  residual sugar  \
0           7.4             0.70         0.00           1.9
1           7.8             0.88         0.00           2.6
2           7.8             0.76         0.04           2.3
3          11.2             0.28         0.56           1.9
4           7.4             0.70         0.00           1.9

     chlorides  free sulfur dioxide  total sulfur dioxide  density  \
0         0.076             11.0             34.0  0.9978
1         0.098             25.0             67.0  0.9968
2         0.092             15.0             54.0  0.9970
3         0.075             17.0             60.0  0.9980
4         0.076             11.0             34.0  0.9978

     pH  sulphates  alcohol  quality
0  3.51         0.56      9.4        5
1  3.20         0.68      9.8        5
2  3.26         0.65      9.8        5
3  3.16         0.58      9.8        6
4  3.51         0.56      9.4        5
```

```
[8]: df = data.copy()
     df.head()
```

```
[8]:      fixed acidity  volatile acidity  citric acid  residual sugar  \
0           7.4           0.70           0.00           1.9
1           7.8           0.88           0.00           2.6
2           7.8           0.76           0.04           2.3
3          11.2           0.28           0.56           1.9
4           7.4           0.70           0.00           1.9

      chlorides  free sulfur dioxide  total sulfur dioxide  density  \
0         0.076           11.0           34.0  0.9978
1         0.098           25.0           67.0  0.9968
2         0.092           15.0           54.0  0.9970
3         0.075           17.0           60.0  0.9980
4         0.076           11.0           34.0  0.9978

      pH  sulphates  alcohol  quality
0  3.51      0.56      9.4        5
1  3.20      0.68      9.8        5
2  3.26      0.65      9.8        5
3  3.16      0.58      9.8        6
4  3.51      0.56      9.4        5
```

```
[9]: df.dtypes
```

```
[9]: 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:

```
[10]: df.shape
```

```
[10]: (1599, 12)
```

Проверим основные статистические характеристики набора данных:

```
[11]: df.describe()
```

```
[11]:      fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000    1599.000000  1599.000000    1599.000000
mean       8.319637       0.527821    0.270976       2.538806
std       1.741096       0.179060    0.194801       1.409928
min       4.600000       0.120000    0.000000       0.900000
25%       7.100000       0.390000    0.090000       1.900000
```

50%	7.900000	0.520000	0.260000	2.200000
75%	9.200000	0.640000	0.420000	2.600000
max	15.900000	1.580000	1.000000	15.500000

	chlorides	free sulfur dioxide	total sulfur dioxide \
count	1599.000000	1599.000000	1599.000000
mean	0.087467	15.874922	46.467792
std	0.047065	10.460157	32.895324
min	0.012000	1.000000	6.000000
25%	0.070000	7.000000	22.000000
50%	0.079000	14.000000	38.000000
75%	0.090000	21.000000	62.000000
max	0.611000	72.000000	289.000000

	density	pH	sulphates	alcohol \
count	1599.000000	1599.000000	1599.000000	1599.000000
mean	0.996747	3.311113	0.658149	10.422983
std	0.001887	0.154386	0.169507	1.065668
min	0.990070	2.740000	0.330000	8.400000
25%	0.995600	3.210000	0.550000	9.500000
50%	0.996750	3.310000	0.620000	10.200000
75%	0.997835	3.400000	0.730000	11.100000
max	1.003690	4.010000	2.000000	14.900000

	quality
count	1599.000000
mean	5.636023
std	0.807569
min	3.000000
25%	5.000000
50%	6.000000
75%	6.000000
max	8.000000

Проверим наличие пропусков в данных:

```
[12]: df.isnull().sum()
```

```
[12]: 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
```

3.2. Разделение данных

Разделим данные на целевой столбец и признаки:

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

```
[14]: print(X.head(), "\n")
      print(y.head())
```

	fixed acidity	volatile acidity	citric acid	residual sugar	\
0	7.4	0.70	0.00	1.9	
1	7.8	0.88	0.00	2.6	
2	7.8	0.76	0.04	2.3	
3	11.2	0.28	0.56	1.9	
4	7.4	0.70	0.00	1.9	

	chlorides	free sulfur dioxide	total sulfur dioxide	pH	\
0	0.076	11.0	34.0	3.51	
1	0.098	25.0	67.0	3.20	
2	0.092	15.0	54.0	3.26	
3	0.075	17.0	60.0	3.16	
4	0.076	11.0	34.0	3.51	

	sulphates	alcohol	quality
0	0.56	9.4	5
1	0.68	9.8	5
2	0.65	9.8	5
3	0.58	9.8	6
4	0.56	9.4	5

0	0.9978
1	0.9968
2	0.9970
3	0.9980
4	0.9978

Name: density, dtype: float64

```
[15]: print(X.shape)
      print(y.shape)
```

(1599, 11)

(1599,)

Предобработаем данные, чтобы методы работали лучше:

```
[16]: columns = X.columns
      scaler = StandardScaler()
      X = scaler.fit_transform(X)
      pd.DataFrame(X, columns=columns).describe()
```

```
[16]:      fixed acidity  volatile acidity  citric acid  \
count      1.599000e+03      1.599000e+03  1.599000e+03
mean      3.554936e-16      1.733031e-16 -8.887339e-17
std       1.000313e+00      1.000313e+00  1.000313e+00
min      -2.137045e+00     -2.278280e+00 -1.391472e+00
25%      -7.007187e-01     -7.699311e-01 -9.293181e-01
50%      -2.410944e-01     -4.368911e-02 -5.636026e-02
75%       5.057952e-01      6.266881e-01  7.652471e-01
max       4.355149e+00      5.877976e+00  3.743574e+00

      residual sugar      chlorides  free sulfur dioxide  \
count      1.599000e+03  1.599000e+03      1.599000e+03
mean      -1.244227e-16  3.821556e-16      -6.221137e-17
std       1.000313e+00  1.000313e+00      1.000313e+00
min      -1.162696e+00 -1.603945e+00      -1.422500e+00
25%      -4.532184e-01 -3.712290e-01      -8.487156e-01
50%      -2.403750e-01 -1.799455e-01      -1.793002e-01
75%       4.341614e-02  5.384542e-02      4.901152e-01
max       9.195681e+00  1.112703e+01      5.367284e+00

      total sulfur dioxide      pH      sulphates  \
count      1.599000e+03  1.599000e+03  1.599000e+03
mean       4.443669e-17  2.861723e-15  6.754377e-16
std       1.000313e+00  1.000313e+00  1.000313e+00
min      -1.230584e+00 -3.700401e+00 -1.936507e+00
25%      -7.440403e-01 -6.551405e-01 -6.382196e-01
50%      -2.574968e-01 -7.212705e-03 -2.251281e-01
75%       4.723184e-01  5.759223e-01  4.240158e-01
max       7.375154e+00  4.528282e+00  7.918677e+00

      alcohol      quality
count  1.599000e+03  1.599000e+03
mean   1.066481e-16  8.887339e-17
std    1.000313e+00  1.000313e+00
min   -1.898919e+00 -3.265165e+00
25%   -8.663789e-01 -7.878226e-01
50%   -2.093081e-01  4.508484e-01
75%    6.354971e-01  4.508484e-01
max    4.202453e+00  2.928190e+00
```

Разделим выборку на тренировочную и тестовую:

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

```
[18]: print(X_train.shape)
      print(X_test.shape)
      print(y_train.shape)
      print(y_test.shape)
```

(1199, 11)

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

3.3. Обучение моделей

Напишем функцию, которая считает метрики построенной модели:

```
[19]: 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)))
```

3.3.1. Случайный лес

Попробуем случайный лес с гиперпараметром $n = 100$:

```
[20]: ran_100 = RandomForestRegressor(n_estimators=100)
      ran_100.fit(X_train, y_train)

[20]: RandomForestRegressor(bootstrap=True, criterion='mse', 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=None,
                             verbose=0, warm_start=False)
```

Проверим метрики построенной модели:

```
[21]: test_model(ran_100)
```

```
mean_absolute_error: 0.0004665994083675076
median_absolute_error: 0.00034904327793133794
r2_score: 0.8848004254396441
```

Видно, что данный метод даже без настройки гиперпараметров уже показывает очень неплохой результат.

3.3.2. Градиентный бустинг

Попробуем градиентный бустинг с гиперпараметром $n = 100$:

```
[22]: gr_100 = GradientBoostingRegressor(n_estimators=100)
      gr_100.fit(X_train, y_train)

[22]: GradientBoostingRegressor(alpha=0.9, 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='auto',
random_state=None, subsample=1.0, tol=0.0001,
validation_fraction=0.1, verbose=0,
warm_start=False)

```

Проверим метрики построенной модели:

```
[23]: test_model(gr_100)
```

```

mean_absolute_error: 0.0004723057776203243
median_absolute_error: 0.00034447129988390834
r2_score: 0.8907447664867403

```

Внезапно градиентный бустинг оказался несколько лучше по сравнению со случайным лесом.

3.4. Подбор гиперпараметра n

3.4.1. Случайный лес

Введем список настраиваемых параметров:

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

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

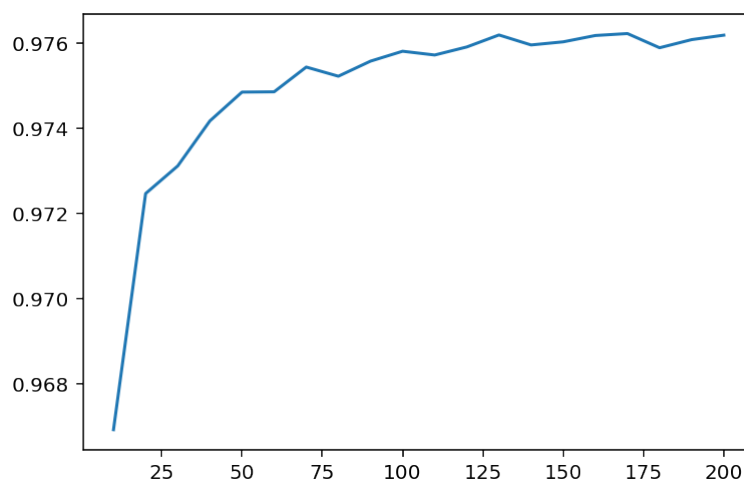
Запустим подбор параметра:

```
[25]: 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_
```

```
[25]: RandomForestRegressor(bootstrap=True, criterion='mse', 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=130,
                           n_jobs=None, oob_score=False, random_state=None,
                           verbose=0, warm_start=False)
```

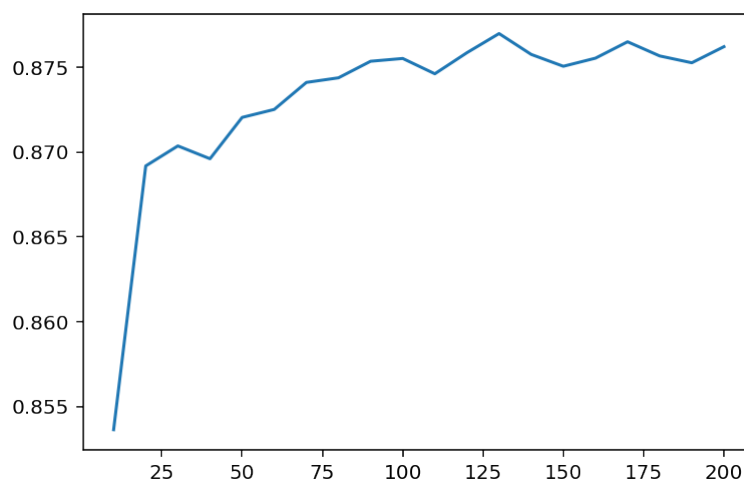
Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

```
[26]: 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:

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



Из-за случайности график немного плавает, но конкретно в данном случае получился чётко выраженный пик с наилучшим результатом.

```
[28]: reg = gs.best_estimator_  
reg.fit(X_train, y_train)  
test_model(reg)
```

```
mean_absolute_error: 0.0004643566289560763  
median_absolute_error: 0.00035339159736763603  
r2_score: 0.8831096743956034
```

Конкретно данная модель оказалась практически такой же, как исходная.

3.4.2. Градиентный бустинг

Список настраиваемых параметров оставим тем же.

```
[29]: tuned_parameters
```

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

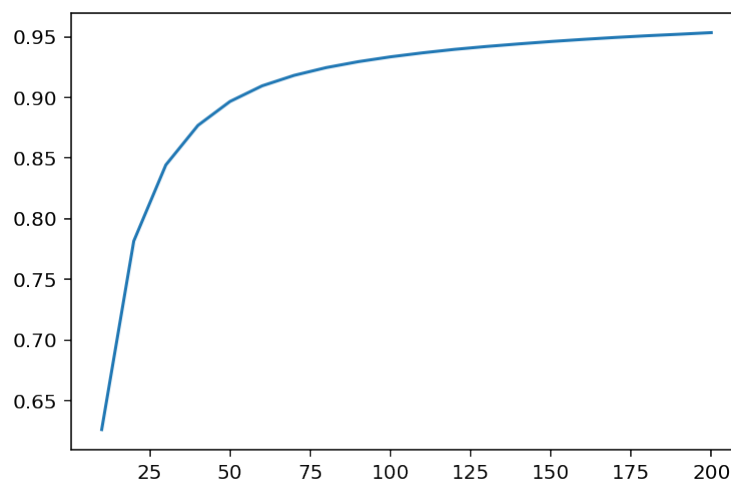
Запустим подбор параметра:

```
[30]: 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_
```

```
[30]: GradientBoostingRegressor(alpha=0.9, 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='auto',
    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 данных:

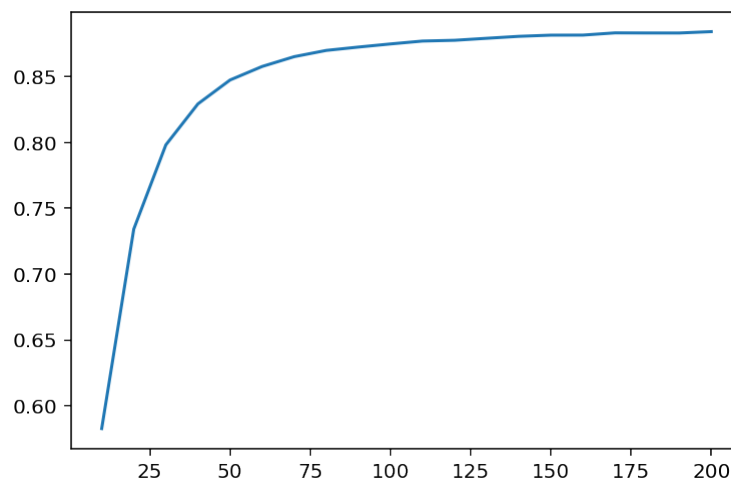
```
[31]: plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



Картина та же: чем больше подмоделей, тем лучше.

На тестовом наборе данных картина ровно та же:

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



Выходит, что чем больше подмоделей, тем лучше. Возможно, что можно использовать ещё больше моделей, но будем считать, что это выходит за рамки лабораторной работы.

```
[33]: reg = gs.best_estimator_  
reg.fit(X_train, y_train)  
test_model(reg)
```

```
mean_absolute_error: 0.00044074456005408017  
median_absolute_error: 0.0003332712029135432  
r2_score: 0.9049735838489492
```

```
[ ]: Модель получилась немного лучше исходной.
```