

Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



“Методы машинного обучения”

ЛАБОРАТОРНАЯ РАБОТА № 6. «Ансамбли моделей машинного обучения»

Студент группы ИУ5-24М

Петропавлов Д.М.

\_\_\_\_\_ Дата

\_\_\_\_\_ Подпись

## Задание:

1. Выберите 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 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]: 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

# Подключаем встроенные графики
%matplotlib inline

# Изменим формат сохранения графиков для улучшения отображения
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

## Подготовка данных.

```
In [2]: #Подключаем данные
data = pd.read_csv('SolarPrediction.csv', sep=",")

In [3]: #Преобразуем временные колонки в соответствующий временной формат:
data["Time"] = (pd
                .to_datetime(data["UNIXTime"], unit="s", utc=True)
                .dt.tz_convert("Pacific/Honolulu")).dt.time

data["TimeSunRise"] = (pd
                      .to_datetime(data["TimeSunRise"],
                                   infer_datetime_format=True)
                      .dt.time)

data["TimeSunSet"] = (pd
                     .to_datetime(data["TimeSunSet"],
                                   infer_datetime_format=True)
                     .dt.time)

data = data.rename({"WindDirection(Degrees)": "WindDirection"},
                  axis=1)
```

```
In [4]: def time_to_second(t):
        return ((datetime.combine(datetime.min, t) - datetime.min)
                .total_seconds())
```

```
In [5]: df = data.copy()

timeInSeconds = df["Time"].map(time_to_second)

sunrise = df["TimeSunRise"].map(time_to_second)
sunset = df["TimeSunSet"].map(time_to_second)
df["DayPart"] = (timeInSeconds - sunrise) / (sunset - sunrise)

df = df.drop(["UNIXTime", "Data", "Time",
             "TimeSunRise", "TimeSunSet"], axis=1)

df.head()
```

```
Out[5]:
```

	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
0	1.21	48	30.46	59	177.39	5.62	1.475602
1	1.21	48	30.46	58	176.78	3.37	1.468588
2	1.23	48	30.46	57	158.75	3.37	1.461713
3	1.21	48	30.46	60	137.71	3.37	1.454653
4	1.17	48	30.46	62	104.95	5.62	1.447778

```
In [6]: df.dtypes
```

```
Out[6]: Radiation      float64
Temperature    int64
Pressure       float64
Humidity       int64
WindDirection  float64
Speed          float64
DayPart        float64
dtype: object
```

```
In [7]: df.shape
```

```
Out[7]: (32686, 7)
```

```
In [8]: # Проверим наличие пустых значений
df.isnull().sum()
```

```
Out[8]: Radiation      0
Temperature    0
Pressure       0
Humidity       0
WindDirection  0
Speed          0
DayPart        0
dtype: int64
```

## Разделение данных.

```
In [9]: X = df.drop("Radiation", axis=1)
y = df["Radiation"]
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

```
Out[9]:
```

	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
count	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04
mean	5.565041e-16	2.904952e-14	1.391260e-17	6.956302e-17	-9.738822e-17	5.217226e-18
std	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00
min	-2.758117e+00	-4.259540e+00	-2.578560e+00	-1.724255e+00	-1.788859e+00	-1.855112e+00
25%	-8.229646e-01	-4.184734e-01	-7.316829e-01	-7.366250e-01	-8.233591e-01	-8.683240e-01
50%	-1.779139e-01	1.302504e-01	3.841388e-01	5.062367e-02	-1.787376e-01	2.279483e-03
75%	6.283995e-01	6.789742e-01	8.458578e-01	4.307058e-01	4.658840e-01	8.682924e-01
max	3.208803e+00	2.508053e+00	1.076717e+00	2.602741e+00	9.814329e+00	1.797910e+00

```
In [10]: #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)

(24514, 6)
(8172, 6)
(24514,)
(8172,)
```

## Обучение моделей

```
In [11]: #функция, 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

```
In [12]: # C гиперпараметром n=100:
ran_100 = RandomForestRegressor(n_estimators=100)
ran_100.fit(X_train, y_train)

Out[12]: 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)

In [13]: test_model(ran_100)

mean_absolute_error: 37.991201309349
median_absolute_error: 0.6180000000000014
r2_score: 0.915709286344976
```

## Градиентный бустинг

```
In [14]: # C zuneprnapamempom n=100:
gr_100 = GradientBoostingRegressor(n_estimators=100)
gr_100.fit(X_train, y_train)

Out[14]: 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)

In [15]: test_model(gr_100)

mean_absolute_error: 58.07682041283236
median_absolute_error: 14.74142199396115
r2 score: 0.8729966247836403
```

### Подбор гиперпараметра $n$

## Случайный лес

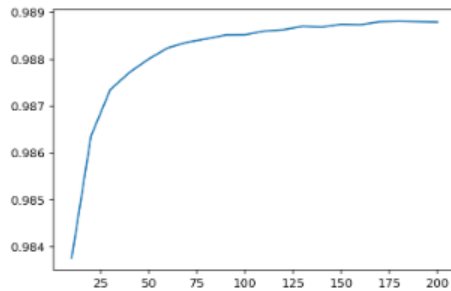
```
In [16]: # Список настраиваемых параметров
param_range = np.arange(10, 201, 10)
tuned_parameters = [{'n_estimators': param_range}]
tuned_parameters

Out[16]: [{'n_estimators': array([ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130,
140, 150, 160, 170, 180, 190, 200])}]
```

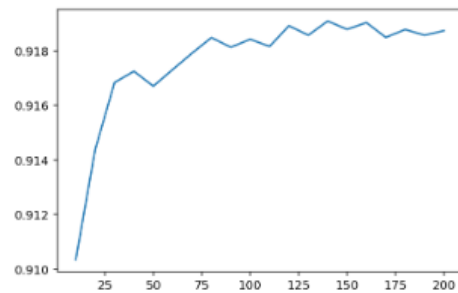
```
In [17]: gs = GridSearchCV(RandomForestRegressor(), tuned_parameters,
                           cv=ShuffleSplit(n_splits=10), scoring="r2",
                           return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

```
Out[17]: 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=140, n_jobs=None, oob_score=False,
                               random_state=None, verbose=0, warm_start=False)
```

```
In [18]: plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



```
In [19]: plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



```
In [20]: reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

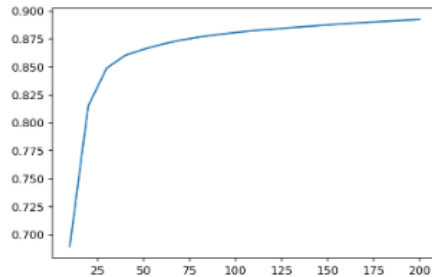
```
mean_absolute_error: 37.70344277498077
median_absolute_error: 0.5943214285714279
r2_score: 0.9163774164575984
```

## Градиентный бустинг

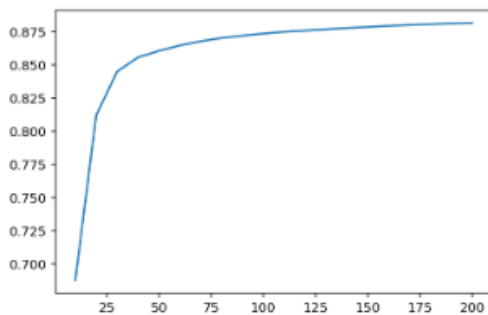
```
In [21]: gs = GridSearchCV(GradientBoostingRegressor(), tuned_parameters,
                           cv=ShuffleSplit(n_splits=10), scoring="r2",
                           return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

```
Out[21]: 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)
```

```
In [22]: plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



```
In [23]: plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



```
In [24]: reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)

mean_absolute_error: 55.851375639767475
median_absolute_error: 14.572172035170793
r2_score: 0.8810189281886804
```

Метод Случайного леса показал неплохие результаты.

Градиентный бустинг оказался хуже случайного леса.

Видно, что при увеличении обучаемых моделей, тем выше точность, однако из-за случайности, график немного плавает. Модель случайного леса оказалась лучше исходной.

С методом градиентного бустинга картина та же: больше моделей -> лучше результаты.