

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

Факультет «Информатика и системы управления»  
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа № 6**  
по дисциплине «Методы машинного обучения»

Тема: «Ансамбли моделей машинного обучения.»

ИСПОЛНИТЕЛЬ:

группа ИУ5-24М

Шапиев М.М.

ФИО

подпись

"\_\_" \_\_\_\_\_ 2020 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

ФИО

подпись

"\_\_" \_\_\_\_\_ 2020 г.

Москва - 2020

---

## 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.

## Реализация задания

### Часть 1. Предварительная подготовка данных

Очевидно, что все эти временные характеристики в таком виде нам не особо интересны. Преобразуем все нечисловые столбцы в числовые. В целом колонка `UNIXTime` нам не интересна, дата скорее интереснее в виде дня в году. Время измерения может быть интересно в двух видах: просто секунды с полуночи, и время, нормализованное относительно рассвета и заката.

```
#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)
```

```
def time_to_second(t):
    return ((datetime.combine(datetime.min, t) - datetime.min)
        .total_seconds())
```

```
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()
```

Результат преобразования данных:

	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

```
df.dtypes
```

```
Radiation      float64
Temperature     int64
Pressure        float64
Humidity        int64
WindDirection   float64
Speed           float64
DayPart         float64
dtype: object
```

```
df.shape
```

```
(32686, 7)
```

Проверим набор данных на наличие пустых значений:

```
# Проверим наличие пустых значений
df.isnull().sum()
```

```
Radiation      0
Temperature     0
Pressure        0
Humidity        0
WindDirection   0
Speed           0
DayPart         0
dtype: int64
```

## Часть 2. Разделение данных.

```
: 1 X = df.drop("Radiation", axis=1)
2 y = df["Radiation"]
3 columns = X.columns
4 scaler = StandardScaler()
5 X = scaler.fit_transform(X)
6 pd.DataFrame(X, columns=columns).describe()
```

```
:
      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  3.841386e-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     3.208603e+00  2.508053e+00  1.076717e+00  2.602741e+00  9.814329e+00  1.797910e+00
```

```
: 1 #Разделим выборку на тренировочную и тестовую
2 X_train, X_test, y_train, y_test = train_test_split(X, y,
3                                                    test_size=0.25, random_state=346705925)
4 print(X_train.shape)
5 print(X_test.shape)
6 print(y_train.shape)
7 print(y_test.shape)
```

(24514, 6)

(8172, 6)

(24514,)

(8172,)

## Часть 3. Обучение модели.

### Случайный лес

```
1 # С гиперпараметром n=100:  
2 ran_100 = RandomForestRegressor(n_estimators=100)  
3 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)
```

```
1 test_model(ran_100)
```

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

Даже без настройки параметров данный метод показал хорошие результаты.

### Градиентный бустинг

```
1 # С гиперпараметром n=100:  
2 gr_100 = GradientBoostingRegressor(n_estimators=100)  
3 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)
```

```
1 test_model(gr_100)
```

```
mean_absolute_error: 58.07682041283236  
median_absolute_error: 14.74142199396115  
r2_score: 0.8729966247836403
```

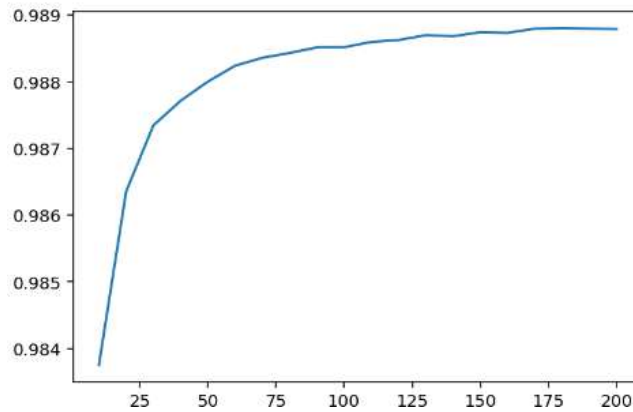
По результатам видно что градиентный бустинг оказался хуже случайного леса

## Часть 4. Подбор гиперпараметра $n$

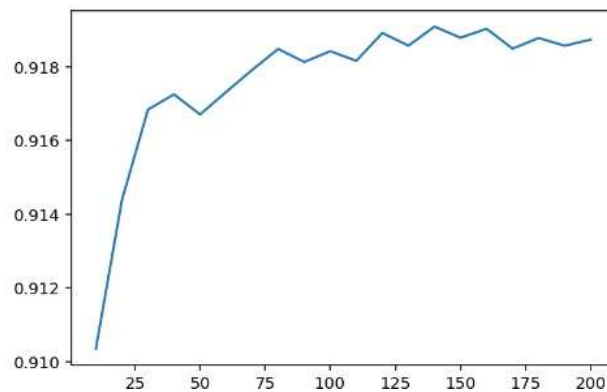
### Случайный лес

Подберём параметры, а потом проверим на тренировочном и тестовом наборе данных:

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



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



Видно что при увеличении обучаемых моделей, тем точность выше, однако из-за случайности график немного плавает, но конкретно в данном случае получился чётко выраженный пик с наилучшим результатом.

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

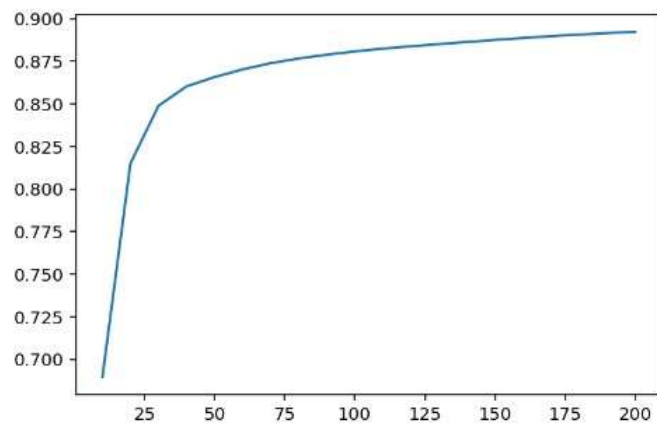
```
mean_absolute_error: 37.70344277498077  
median_absolute_error: 0.5943214285714279  
r2_score: 0.9163774164575984
```

Данная модель оказалась лучше, чем исходная.

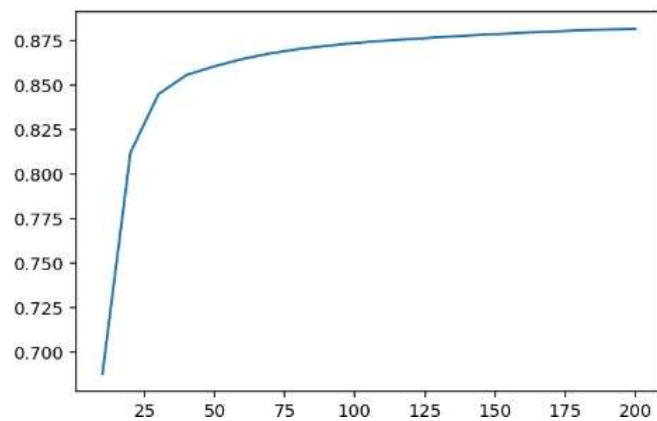
## Градиентный бустинг

Подберём параметры, а потом проверим на тренировочном и тестовом наборе данных:

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



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



Картина та же: чем больше моделей, тем результаты лучше.

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

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

Данная модель так же оказалась лучше, чем исходная.