МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

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

ОТЧЕТ

Домашнее задание №__1__
по дисциплине «Методы машинного обучения»

ИСПОЛНИТЕЛЬ: Егоров С.А.
ФИО
группа ИУ5-22М _____
подпись
"__"_____2020 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.Е.
ФИО
_____
подпись
"__"_____2020 г.

Москва - 2020

Задание

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

Реализация задания

Выбор набора 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) и использовался в соревновании Space Apps Moscow 2017 в категории «You are my Sunshine» для построения приложения для предсказания мощности солнечного излучения и планирования работы исследовательской станции. Данный набор данных доступен по следующему адресу: https://www.kaggle.com/dronio/SolarEnergy.

Текстовое описание набора данных

Выбранный набор данных состоит из одного файла SolarPrediction.csv, содержащего все данные датасета. Данный файл содержит следующие колонки:

- UNIXTime — временная метка измерения в формате UNIX;
- Data — дата измерения;
- Time — время измерения (в местной временной зоне);

- Radiation — солнечное излучение ( );
- Temperature — температура (°F);
- Pressure — атмосферное давление (дюймов ртутного столба);
- Humidity — относительная влажность (%);
- WindDirection(Degrees) — направление ветра (°);
- Speed — скорость ветра (миль/ч);
- TimeSunRise — время восхода (в местной временной зоне);
- TimeSunSet — время заката (в местной временной зоне).

Постановка задачи и предварительный анализ набора данных

Очевидно, что данный набор данных предполагает задачу регрессии, а именно предсказание колонки Radiation — мощности солнечного излучения. При этом:

- Колонка UNIXTime сама по себе довольно бесполезна, так как просто монотонно растёт с течением времени, не давая какую-либо информацию для модели машинного обучения. Вместе с тем, колонка Time может быть довольно интересной, особенно вместе с колонками TimeSunRise и TimeSunSet, так как вместе они показывают положение солнца на небе и точно задают возможный максимум солнечной энергии.
- Колонка Data могла бы быть полезна, если бы данные были за больший промежуток времени (например, несколько лет), и отражала бы сезонность солнечного излучения. К сожалению, в нашем случае она практически полностью бесполезна.
- Остальные колоки предоставляют данные, которые теоретически могут показывать, сколько именно солнечной энергии доходит до поверхности, то есть по факту по ним необходимо предсказывать облачность.

Часть 1. Предварительная подготовка данных

```
#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)
```

Часть 2. Разведочный анализ

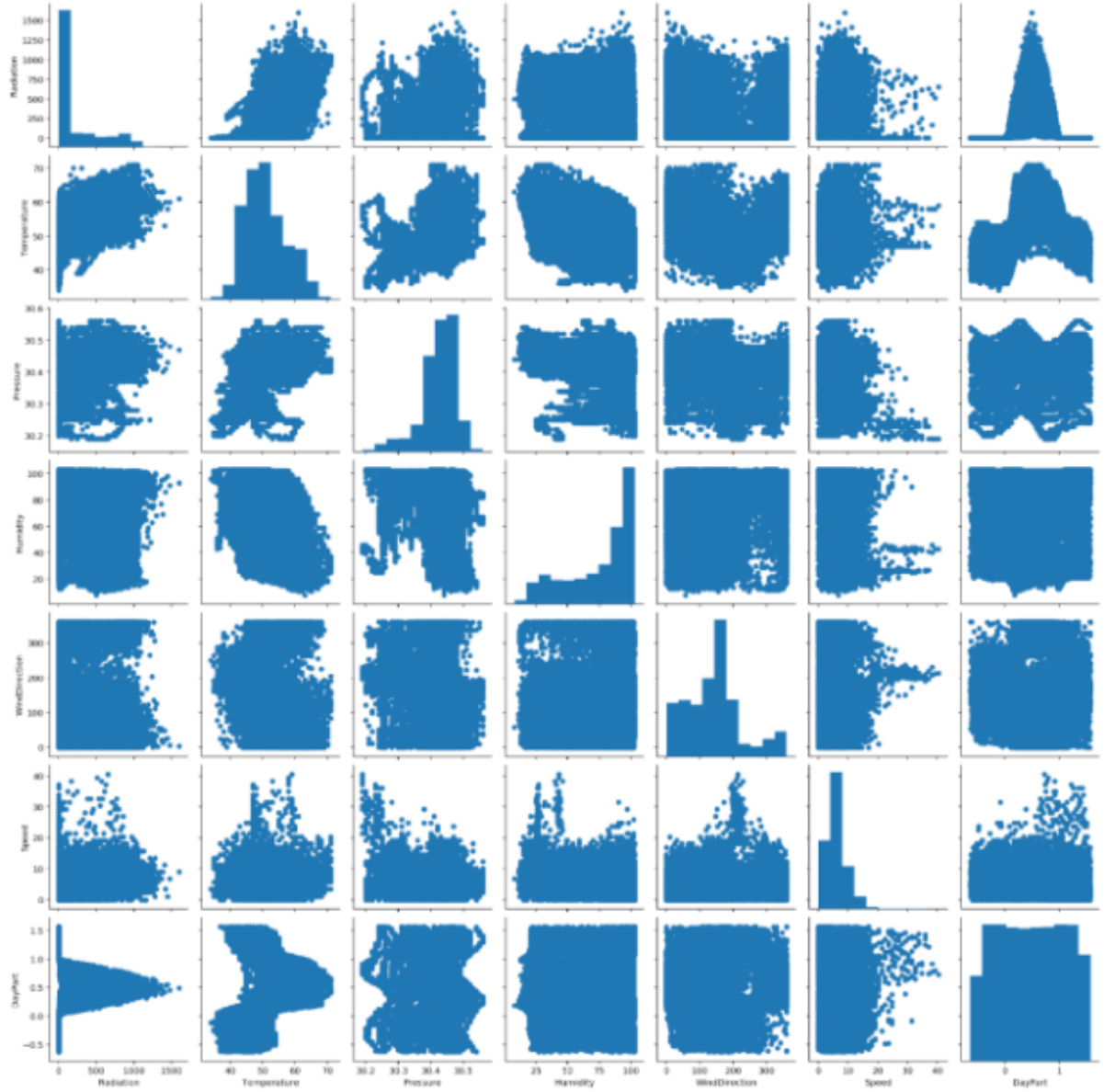
```
# Основные статистические характеристики
df.describe()
```

	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
count	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000
mean	207.124697	51.103255	30.422879	75.016307	143.489821	6.243869	0.482959
std	315.916387	6.201157	0.054673	25.990219	83.167500	3.490474	0.602432
min	1.110000	34.000000	30.190000	8.000000	0.090000	0.000000	-0.634602
25%	1.230000	46.000000	30.400000	56.000000	82.227500	3.370000	-0.040139
50%	2.660000	50.000000	30.430000	85.000000	147.700000	5.620000	0.484332
75%	354.235000	55.000000	30.460000	97.000000	179.310000	7.870000	1.006038
max	1601.260000	71.000000	30.560000	103.000000	359.950000	40.500000	1.566061

```
# Проверка на наличие пропусков
df.isnull().sum()
```

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

```
#Парные диаграммы
sns.pairplot(df, plot_kws=dict(linewidth=0));
```



Видно, что зависимости между колонками весьма сложные и в большинстве своём нелинейные. Какого-то показателя, точно определяющего мощность излучения, не наблюдается. Вместе с тем чётко видно, что время суток ограничивает мощность излучения сверху, что вполне может быть полезно для модели машинного обучения.

Часть 3. Корреляционный анализ

```
1 sns.heatmap(df.corr(), annot=True, fmt=".2f");
```

Видно, что мощность солнечного излучения заметно коррелирует с температурой, что было показано выше с помощью парного графика. Остальные признаки коррелируют друг с другом довольно слабо.

Часть 4. Формирование обучающей и тестовой выборок

```
1 #Разделим на целевой столбец и признаки
2 X = df.drop("Radiation", axis=1)
3 y = df["Radiation"]
4
5 print(X.head(), "\n")
6 print(y.head())
```

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

0    1.21
1    1.21
2    1.23
3    1.21
4    1.17
Name: Radiation, dtype: float64
```

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(X, y,
4                                 test_size=0.25, random_state=346705925)
```

```
1 print(X_train.shape)
2 print(X_test.shape)
3 print(y_train.shape)
4 print(y_test.shape)
```

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

Часть 5. Выбор метрик

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import r2_score

#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)))
```

Очевидно, что все эти метрики подходят для задачи регрессии. При этом средняя абсолютная ошибка (mean_absolute_error) будет показывать, насколько в среднем мы ошибаемся, медианная абсолютная ошибка (median_absolute_error) — насколько мы ошибаемся на половине выборки, а коэффициент детерминации (r2_score) хорош тем, что он показывает качество модели машинного обучения в задачи регрессии без сравнения с другими моделями.

В качестве моделей машинного обучения выберем хорошо показавшие себя в лабораторных работах модели:

- Метод К ближайших соседей (KNeighborsRegressor)
- Дерево решений (DecisionTreeRegressor)
- Случайный лес (RandomForestRegressor)

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

Часть 6. Построение базового решения

Метод k ближайших соседей

```
#С гиперпараметром k=5
knn_5 = KNeighborsRegressor(n_neighbors=5)
knn_5.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

```
test_model(knn_5)
```

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.017000000000004
r2_score: 0.8677873476991447
```

Данный метод даже без настройки гиперпараметров уже показывает очень неплохой результат.

Дерево решений

```
#С неограниченной глубиной
dt_none = DecisionTreeRegressor(max_depth=None)
dt_none.fit(X_train, y_train)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      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, presort='deprecated',
                      random_state=None, splitter='best')
```

```
test_model(dt_none)
```

```
mean_absolute_error: 49.394926578560934
median_absolute_error: 0.7249999999999523
r2_score: 0.837394846291527
```

Данный метод также без настройки гиперпараметров показывает приличный результат.

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)
```

```
test_model(ran_100)
```

```
mean_absolute_error: 37.9442242657856
median_absolute_error: 0.6369000000000611
r2_score: 0.9154556610041328
```

Данный метод даже без настройки гиперпараметров показывает очень хороший результат.

Часть 7. Подбор гиперпараметров

Для подбора гиперпараметров будем использовать следующие библиотеки:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
```

Метод k ближайших соседей

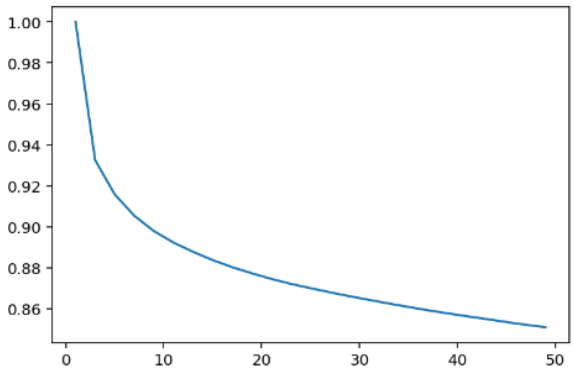
Подберём гиперпараметр K:

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

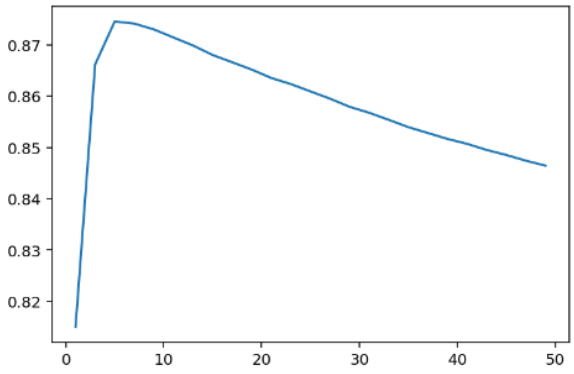
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

Проверим результаты при разных значения гиперпараметра K 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 наилучший результат достигается при k=5.

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

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.017000000000004
r2_score: 0.8677873476991447
```

Получили такой же результат что и на исходной модели.

Дерево решений

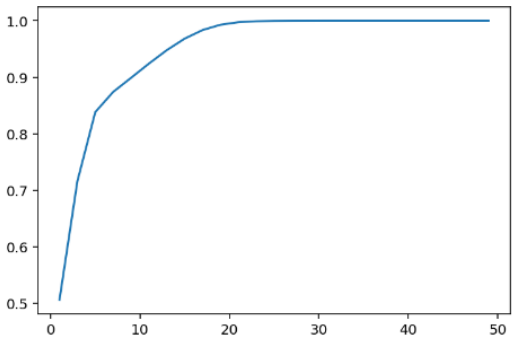
Подберём параметр «Глубина дерева решений»:

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

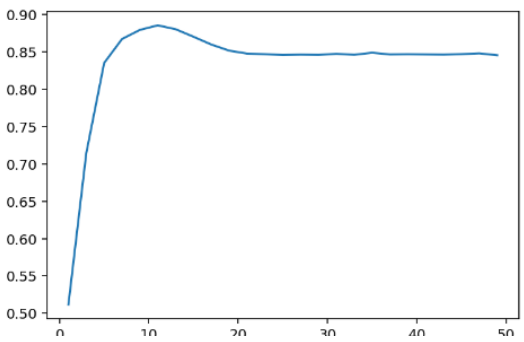
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=11,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

Проверим результаты при разных значениях гиперпараметра на тренировочном и тестовом наборе данных:

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




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



На графике чётко видно, что модель сначала работает хорошо, а потом начинает переобучаться на тренировочной выборке.

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

```
mean_absolute_error: 48.525115383431945
median_absolute_error: 0.8904186804416244
r2_score: 0.869185719078307
```

Конкретная модель оказалась лучше, чем исходная.

Случайный лес

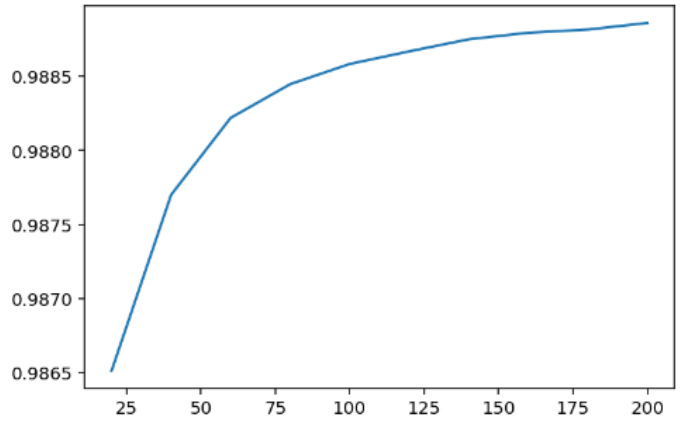
Подберём гиперпараметр n:

```
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_
```

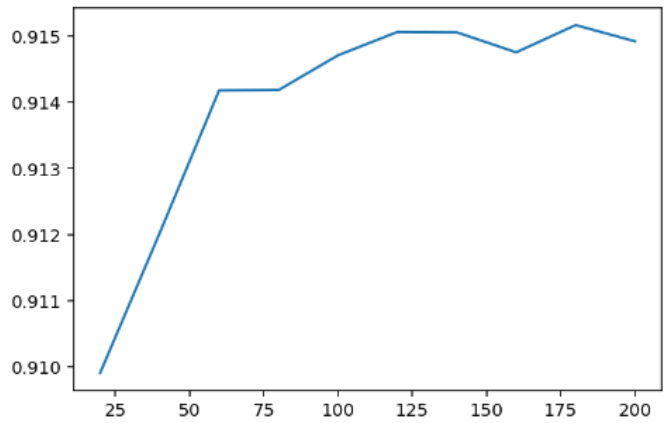
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=180, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Проверим результаты при разных значениях гиперпараметра на тренировочном и тестовом наборе данных:

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




```
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.

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

```
mean_absolute_error: 37.82005191167673
median_absolute_error: 0.6110833333333303
r2_score: 0.9162422360120536
```

Данная модель оказалась чуть-чуть лучше, чем исходная.

Выводы

Все построенные модели обладают очень хорошими показателями. Ансамблевая модель при этом обладает наилучшими характеристиками. Таким образом для дальнейшей работы стоит использовать именно ее.