

Домашняя работа
по дисциплине
«Методы машинного обучения»

Выполнил:
студент группы ИУ5-24М
Лещев А. О.

1. Задание

Требуется выполнить следующие действия [1]:

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.

2. Ход выполнения работы

2.1. 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 [2, 3]. Данный набор данных доступен по следующему адресу: <https://www.kaggle.com/dronio/SolarEnergy>.

2.1.1. 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 (Вт/м^2);
- `Temperature` — температура ($^{\circ}\text{F}$);
- `Pressure` — атмосферное давление (дюймов ртутного столба);
- `Humidity` — относительная влажность (%);
- `WindDirection(Degrees)` — направление ветра ($^{\circ}$);
- `Speed` — скорость ветра (миль/ч);
- `TimeSunRise` — время восхода (в местной временной зоне);
- `TimeSunSet` — время заката (в местной временной зоне).

2.1.2. 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.

2.2. Проведение разведочного анализа данных

Подключим все необходимые библиотеки:

```
[1]: from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

Настроим отображение графиков [4, 5]:

```
[2]: # Enable inline plots
%matplotlib inline

# Set plot style
sns.set(style="ticks")

# 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 [6]:

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

2.2.1. Предварительная подготовка данных

Загрузим описанный выше набор данных:

```
[4]: data = pd.read_csv("./SolarPrediction.csv")
```

Преобразуем временные колонки в соответствующий временной формат:

```
[5]: 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)
```

Проверим полученные типы:

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

```
[6]: UNIXTime          int64
Data                  object
Time                  object
Radiation             float64
Temperature            int64
Pressure              float64
Humidity              int64
WindDirection         float64
Speed                 float64
TimeSunRise           object
TimeSunSet            object
dtype: object
```

Посмотрим на данные в данном наборе данных:

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

```
[7]:   UNIXTime          Data    Time  Radiation  \
0  1475229326  9/29/2016 12:00:00 AM  23:55:26    1.21
1  1475229023  9/29/2016 12:00:00 AM  23:50:23    1.21
```

2	1475228726	9/29/2016	12:00:00	AM	23:45:26	1.23
3	1475228421	9/29/2016	12:00:00	AM	23:40:21	1.21
4	1475228124	9/29/2016	12:00:00	AM	23:35:24	1.17

	Temperature	Pressure	Humidity	WindDirection	Speed	\
0	48	30.46	59	177.39	5.62	
1	48	30.46	58	176.78	3.37	
2	48	30.46	57	158.75	3.37	
3	48	30.46	60	137.71	3.37	
4	48	30.46	62	104.95	5.62	

	TimeSunRise	TimeSunSet
0	06:13:00	18:13:00
1	06:13:00	18:13:00
2	06:13:00	18:13:00
3	06:13:00	18:13:00
4	06:13:00	18:13:00

Очевидно, что все эти временные характеристики в таком виде нам не особо интересны. Преобразуем все нечисловые столбцы в числовые. Для преобразования времени в секунды используем следующий метод [7]:

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

```
[9]: 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()
```

```
[9]:
```

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

	DayPart
0	1.475602
1	1.468588
2	1.461713
3	1.454653

4 1.447778

```
[10]: df.dtypes
```

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

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

```
[11]: df.shape
```

```
[11]: (32686, 7)
```

Проверим основные статистические характеристики набора данных:

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

```
[12]:
```

	Radiation	Temperature	Pressure	Humidity \
count	32686.000000	32686.000000	32686.000000	32686.000000
mean	207.124697	51.103255	30.422879	75.016307
std	315.916387	6.201157	0.054673	25.990219
min	1.110000	34.000000	30.190000	8.000000
25%	1.230000	46.000000	30.400000	56.000000
50%	2.660000	50.000000	30.430000	85.000000
75%	354.235000	55.000000	30.460000	97.000000
max	1601.260000	71.000000	30.560000	103.000000

	WindDirection	Speed	DayPart
count	32686.000000	32686.000000	32686.000000
mean	143.489821	6.243869	0.482959
std	83.167500	3.490474	0.602432
min	0.090000	0.000000	-0.634602
25%	82.227500	3.370000	-0.040139
50%	147.700000	5.620000	0.484332
75%	179.310000	7.870000	1.006038
max	359.950000	40.500000	1.566061

Проверим наличие пропусков в данных:

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

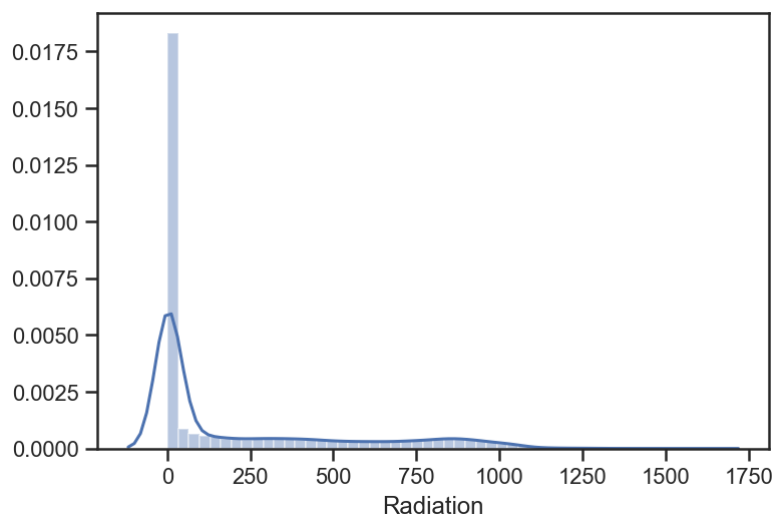
```
[13]: Radiation      0
      Temperature  0
      Pressure     0
      Humidity     0
      WindDirection 0
      Speed        0
      DayPart      0
```

dtype: int64

2.2.2. Визуальное исследование датасета

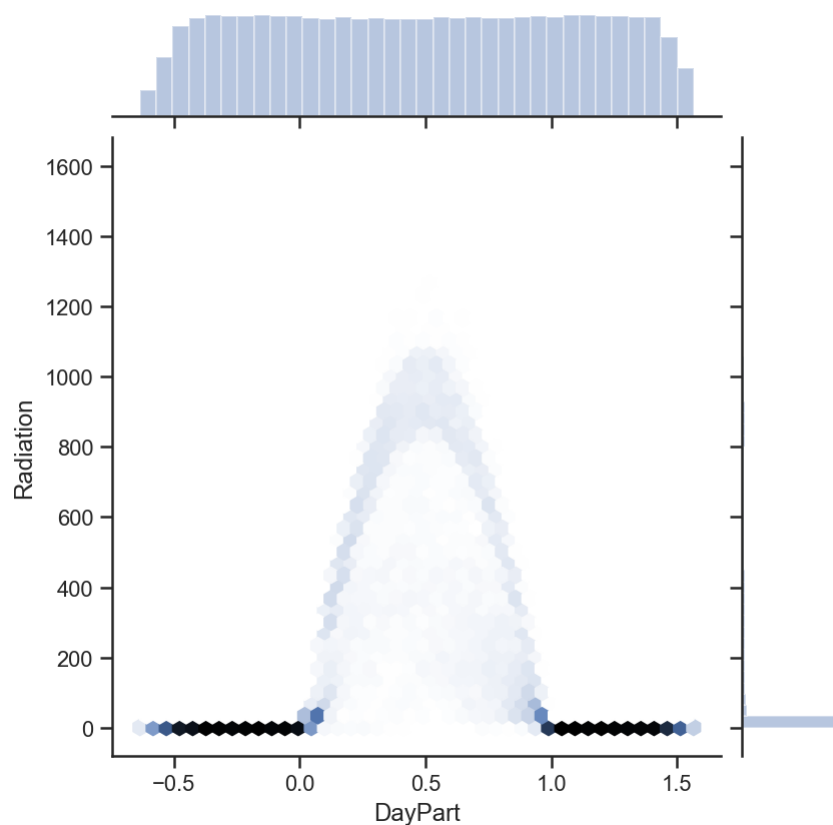
Оценим распределение целевого признака — мощности солнечного излучения:

```
[14]: sns.distplot(df["Radiation"]);
```



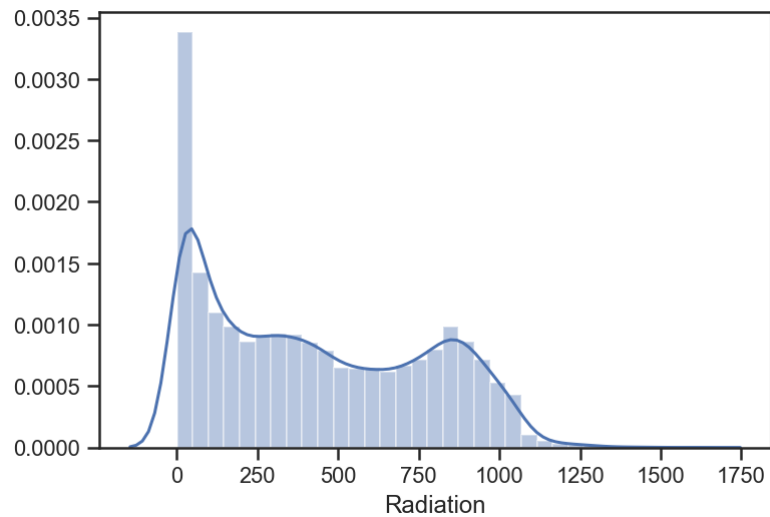
Видно, что имеется большой перевес в пользу практически нулевого излучения. Оценим, насколько мощность солнечного излучения зависит от наличия солнца на небе:

```
[15]: sns.jointplot(x="DayPart", y="Radiation", data=df, kind="hex");
```



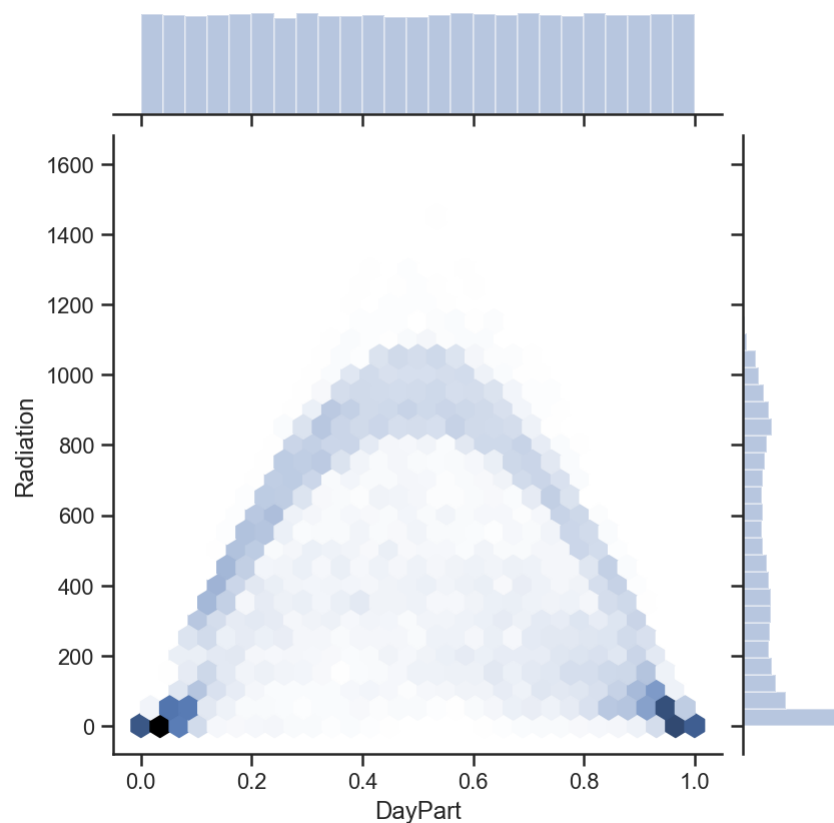
Видно, что если солнца нет на небе, то мощность солнечного излучения стремится к нулю. Посмотрим на распределение мощности излучения в течение дня:

```
[16]: dfd = df[(df["DayPart"] >= 0) & (df["DayPart"] <= 1)]  
sns.distplot(dfd["Radiation"]);
```



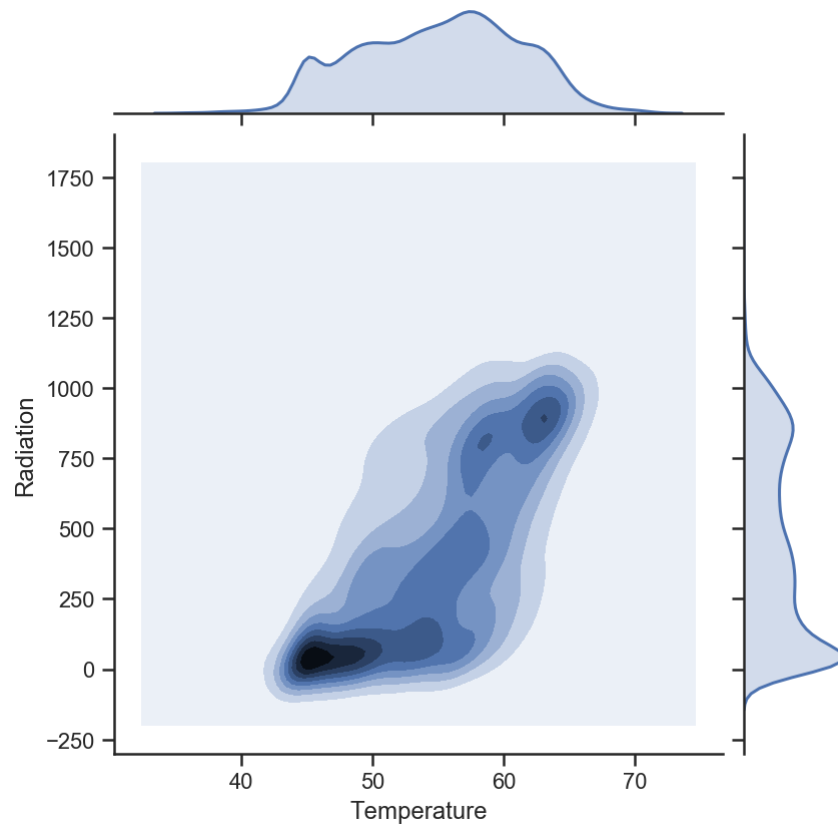
Теперь оценить влияние времени дня на мощность солнечного излучения будет заметно проще:

```
[17]: sns.jointplot(x="DayPart", y="Radiation", data=dfd, kind="hex");
```



Посмотрим также на зависимость мощности солнечного излучения от температуры:

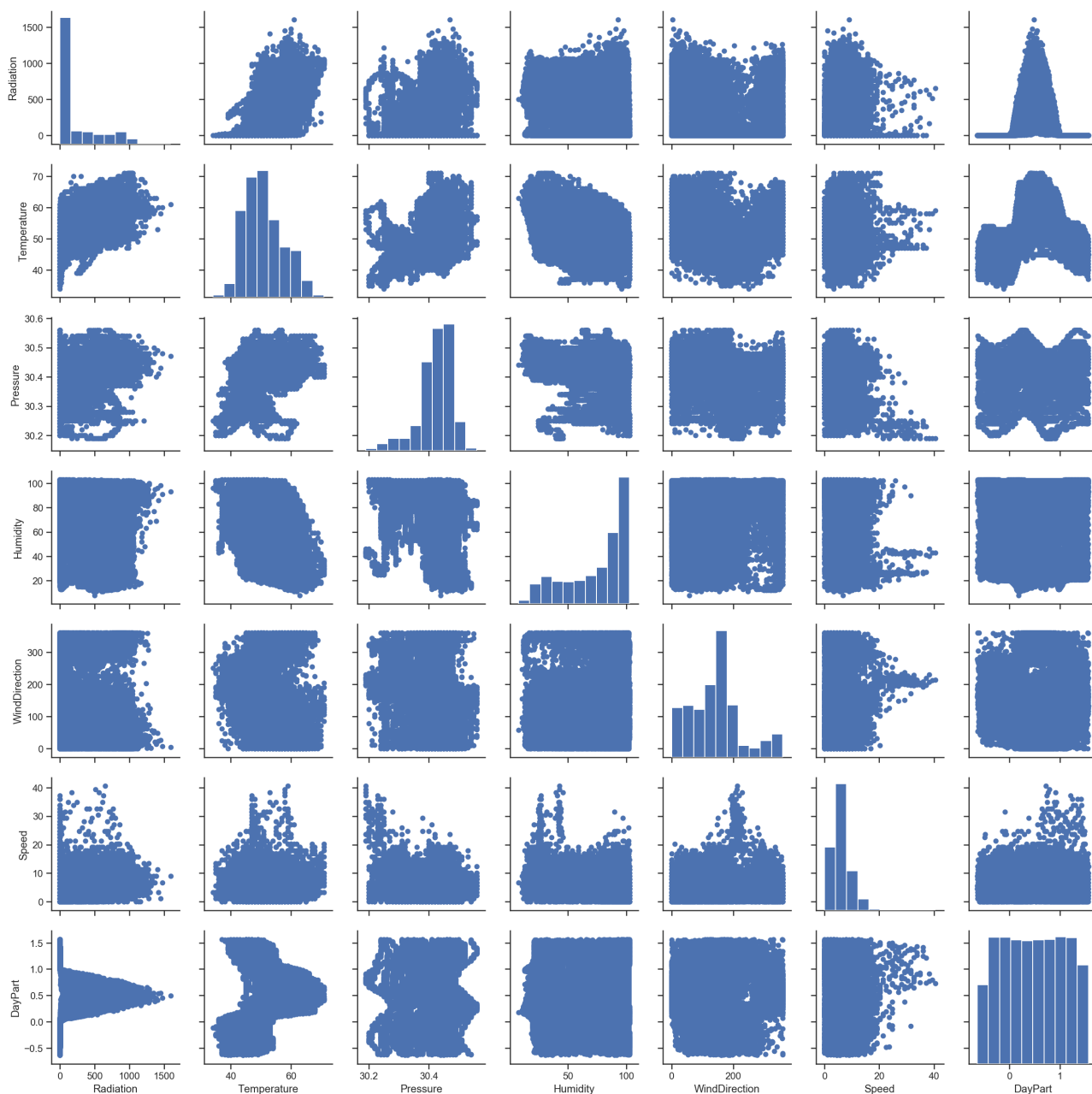
```
[18]: sns.jointplot(x="Temperature", y="Radiation", data=dfd, kind="kde");
```



Видно, что некоторая зависимость определённо есть, но не настолько большая, насколько хотелось бы. Возможно на большей выборке эта зависимость стала бы ещё менее заметной.

Построим парные диаграммы по всем показателям по исходному набору данных:

```
[19]: 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.

2.2.3. Корреляционный анализ

Построим корреляционную матрицу по всему набору данных:

```
[20]: df.corr()
```

```
[20]:
```

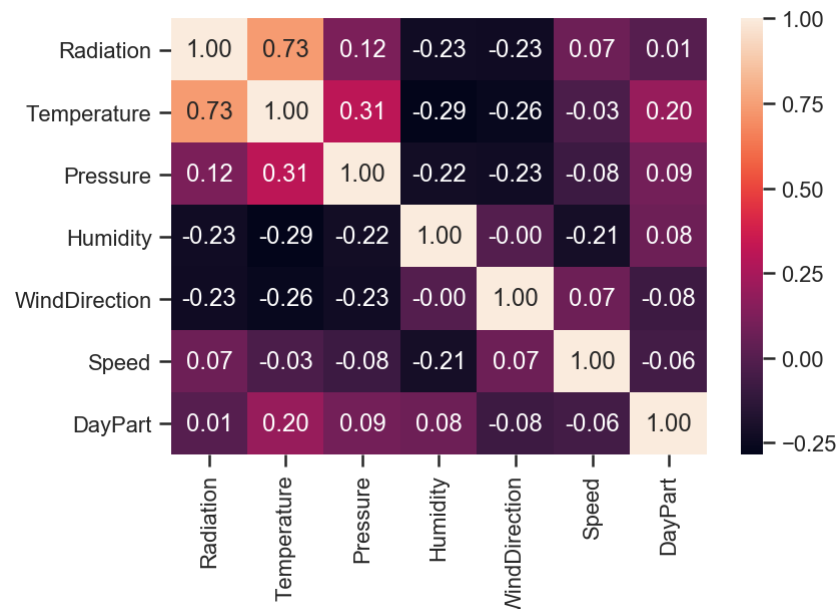
	Radiation	Temperature	Pressure	Humidity	\
Radiation	1.000000	0.734955	0.119016	-0.226171	
Temperature	0.734955	1.000000	0.311173	-0.285055	
Pressure	0.119016	0.311173	1.000000	-0.223973	

Humidity	-0.226171	-0.285055	-0.223973	1.000000
WindDirection	-0.230324	-0.259421	-0.229010	-0.001833
Speed	0.073627	-0.031458	-0.083639	-0.211624
DayPart	0.005980	0.198520	0.094403	0.075513

	WindDirection	Speed	DayPart
Radiation	-0.230324	0.073627	0.005980
Temperature	-0.259421	-0.031458	0.198520
Pressure	-0.229010	-0.083639	0.094403
Humidity	-0.001833	-0.211624	0.075513
WindDirection	1.000000	0.073092	-0.078130
Speed	0.073092	1.000000	-0.056095
DayPart	-0.078130	-0.056095	1.000000

Визуализируем корреляционную матрицу с помощью тепловой карты:

```
[21]: 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. Построению моделей машинного обучения ничего не мешает, но насколько хорошо они будут работать — вопрос открытый.

2.3. Подготовка данных для обучения моделей

Разделим данные на целевой столбец и признаки:

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

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

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

```
(32686, 6)
(32686,)
```

Предобработаем данные, чтобы методы работали лучше:

```
[25]: from sklearn.preprocessing import StandardScaler

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

```
[25]:
```

	Temperature	Pressure	Humidity	WindDirection \
count	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
std	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
25%	-8.229646e-01	-4.184734e-01	-7.316829e-01	-7.366250e-01
50%	-1.779139e-01	1.302504e-01	3.841386e-01	5.062367e-02
75%	6.283995e-01	6.789742e-01	8.458578e-01	4.307058e-01
max	3.208603e+00	2.508053e+00	1.076717e+00	2.602741e+00

	Speed	DayPart
count	3.268600e+04	3.268600e+04
mean	-9.738822e-17	5.217226e-18
std	1.000015e+00	1.000015e+00
min	-1.788859e+00	-1.855112e+00
25%	-8.233591e-01	-8.683240e-01
50%	-1.787376e-01	2.279483e-03
75%	4.658840e-01	8.682924e-01
max	9.814329e+00	1.797910e+00

2.4. Выбор метрик

Напишем функцию, которая считает метрики построенной модели:

```
[26]: from sklearn.metrics import mean_absolute_error
      from sklearn.metrics import median_absolute_error
      from sklearn.metrics import r2_score

      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 R^2 (`r2_score`) хорош тем, что он показывает качество модели машинного обучения в задачи регрессии без сравнения с другими моделями.

2.5. Выбор моделей

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

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

```
[27]: from sklearn.neighbors import KNeighborsRegressor
      from sklearn.tree import DecisionTreeRegressor
      from sklearn.ensemble import RandomForestRegressor
```

2.6. Формирование обучающей и тестовой выборки

Разделим выборку на обучающую и тестовую:

```
[28]: from sklearn.model_selection import train_test_split

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

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

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

2.7. Построение базового решения

2.8. Метод k ближайших соседей

Попробуем метод k ближайших соседей с гиперпараметром $k = 5$:

```
[30]: knn_5 = KNeighborsRegressor(n_neighbors=5)
      knn_5.fit(X_train, y_train)
```

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

Проверим метрики построенной модели:

```
[31]: test_model(knn_5)
```

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.0170000000000004
r2_score: 0.8677873476991447
```

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

2.9. Дерево решений

Попробуем дерево решений с неограниченной глубиной дерева:

```
[32]: dt_none = DecisionTreeRegressor(max_depth=None)
      dt_none.fit(X_train, y_train)
```

```
[32]: DecisionTreeRegressor(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=False, random_state=None, splitter='best')
```

Проверим метрики построенной модели:

```
[33]: test_model(dt_none)
```

```
mean_absolute_error: 50.31291483113069
median_absolute_error: 0.72499999999999659
r2_score: 0.8297706825392527
```

Видно, что данный метод также без настройки гиперпараметров показывает приличный результат.

2.9.1. Случайный лес

Попробуем случайный лес с гиперпараметром $n = 100$:

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

```
[34]: 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:

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

```
mean_absolute_error: 37.503140332843856
median_absolute_error: 0.5816499999999999
r2_score: 0.917512922249891
```

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

2.10. Подбор гиперпараметров

```
[36]: from sklearn.model_selection import GridSearchCV
      from sklearn.model_selection import ShuffleSplit
```

2.10.1. Метод k ближайших соседей

Введем список настраиваемых параметров:

```
[37]: param_range = np.arange(1, 50, 2)
      tuned_parameters = [{'n_neighbors': param_range}]
      tuned_parameters
```

```
[37]: [{'n_neighbors': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23,
                             25, 27,
                             29, 31, 33,
                             35, 37, 39, 41, 43, 45, 47, 49])}]
```

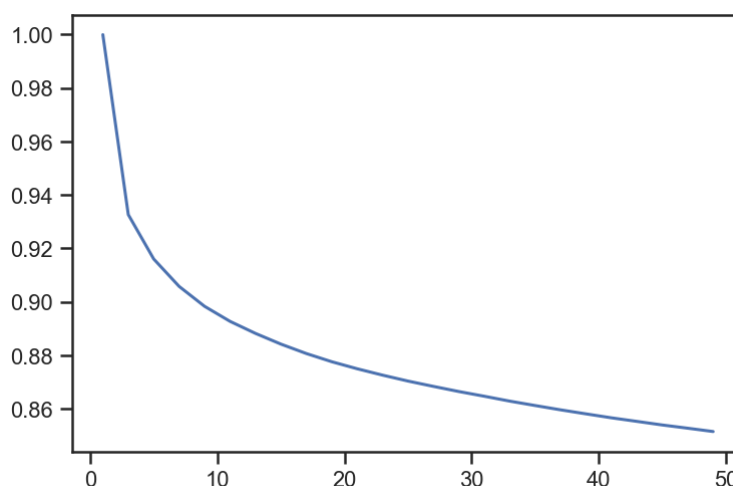
Запустим подбор параметра:

```
[38]: 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_
```

```
[38]: KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                       metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                       weights='uniform')
```

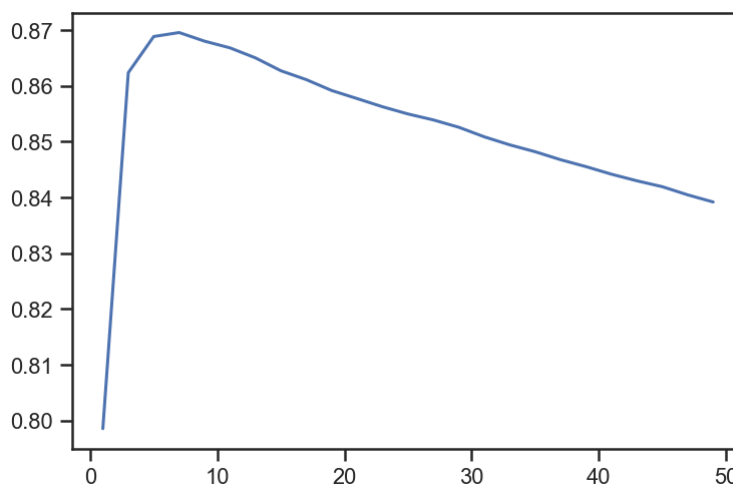
Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

```
[39]: 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:

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



Видно, что наилучший результат достигается при $k = 7$.

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

```
mean_absolute_error: 56.07154831829942  
median_absolute_error: 4.7735714285714295  
r2_score: 0.8687906728428422
```

Сравним с исходной моделью:


```
[42]: test_model(knn_5)
```

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.0170000000000004
r2_score: 0.8677873476991447
```

Здесь получили чуть-чуть больший коэффициент детерминации, но незначительно просели по остальным показателям. Так что делаем вывод, что коэффициент детерминации сам по себе не является идеальной метрикой, и даёт лишь общее представление о качестве модели.

2.10.2. Дерево решений

Введем список настраиваемых параметров:

```
[43]: param_range = np.arange(1, 50, 2)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

```
[43]: [{'max_depth': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27,
    29, 31, 33,
    35, 37, 39, 41, 43, 45, 47, 49])}]
```

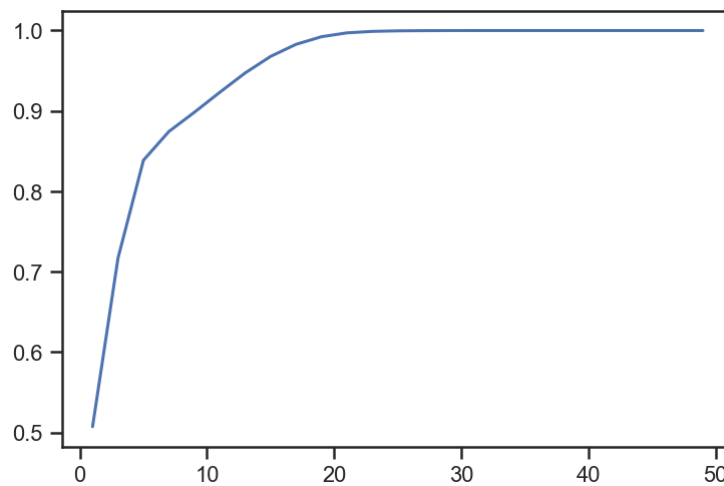
Запустим подбор параметра:

```
[44]: 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_
```

```
[44]: DecisionTreeRegressor(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=False, random_state=None, splitter='best')
```

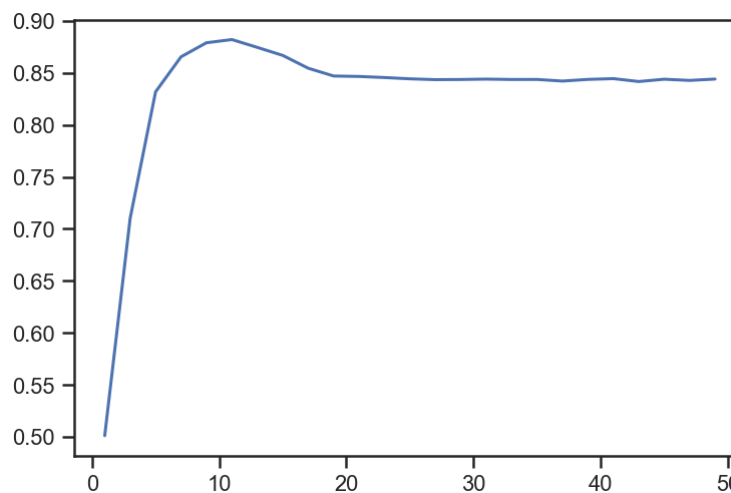
Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

```
[45]: 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:

```
[46]: 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.

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

```
mean_absolute_error: 48.51672010318737  
median_absolute_error: 0.8996284533171739  
r2_score: 0.8698555775877016
```

Сравним с исходной моделью:

```
[48]: test_model(dt_none)
```

```
mean_absolute_error: 50.31291483113069
median_absolute_error: 0.7249999999999659
r2_score: 0.8297706825392527
```

Конкретно данная модель оказалась немного лучше, чем исходная.

2.10.3. Случайный лес

Введем список настраиваемых параметров:

```
[49]: param_range = np.arange(20, 201, 20)
      tuned_parameters = [{'n_estimators': param_range}]
      tuned_parameters
```

```
[49]: [{'n_estimators': array([ 20,  40,  60,  80, 100, 120, 140, 160, 180,
    ↪ 200])}]
```

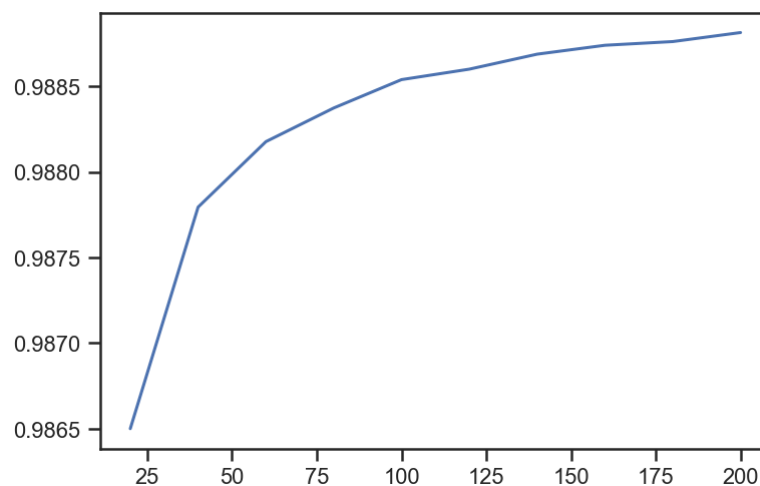
Запустим подбор параметра:

```
[50]: 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_
```

```
[50]: 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=160,
                           n_jobs=None, oob_score=False, random_state=None,
                           verbose=0, warm_start=False)
```

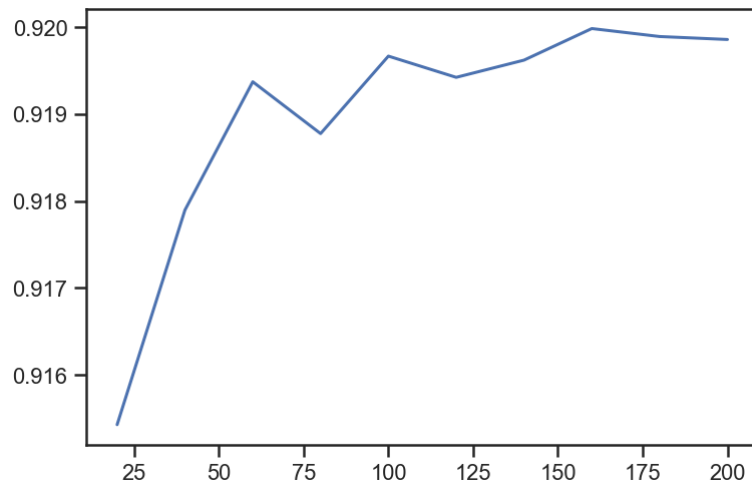
Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

```
[51]: 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:

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



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

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

```
mean_absolute_error: 37.83773731185756  
median_absolute_error: 0.6211875000000019  
r2_score: 0.916153539481388
```

Сравним с исходной моделью:

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

```
mean_absolute_error: 37.503140332843856  
median_absolute_error: 0.5816499999999999  
r2_score: 0.917512922249891
```

Данная модель также оказалась лишь немного лучше, чем исходная.

3. Выводы

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

Список литературы

- [1] Гапанюк Ю. Е. Домашнее задание по дисциплине «Методы машинного обучения» [Электронный ресурс] // GitHub. — 2019. — Режим доступа: https://github.com/ugaryanyuk/ml_course/wiki/MMO_DZ (дата обращения: 06.05.2019).
- [2] You are my Sunshine [Electronic resource] // Space Apps Challenge. — 2017. — Access mode: <https://2017.spaceappschallenge.org/challenges/earth-and-us/you-are-my-sunshine/details> (online; accessed: 22.02.2019).
- [3] dronio. Solar Radiation Prediction [Electronic resource] // Kaggle. — 2017. — Access mode: <https://www.kaggle.com/dronio/SolarEnergy> (online; accessed: 18.02.2019).
- [4] Team The IPython Development. IPython 7.3.0 Documentation [Electronic resource] // Read the Docs. — 2019. — Access mode: <https://ipython.readthedocs.io/en/stable/> (online; accessed: 20.02.2019).
- [5] Waskom M. seaborn 0.9.0 documentation [Electronic resource] // PyData. — 2018. — Access mode: <https://seaborn.pydata.org/> (online; accessed: 20.02.2019).
- [6] pandas 0.24.1 documentation [Electronic resource] // PyData. — 2019. — Access mode: <http://pandas.pydata.org/pandas-docs/stable/> (online; accessed: 20.02.2019).
- [7] Chrétien M. Convert datetime.time to seconds [Electronic resource] // Stack Overflow. — 2017. — Access mode: <https://stackoverflow.com/a/44823381> (online; accessed: 20.02.2019).