Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №5
по дисциплине
«Методы машинного обучения»
на тему
«Линейные модели, SVM и деревья решений»

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

Москва — 2019 г.

1. Цель лабораторной работы

Изучить линейные модели, SVM и деревья решений [1].

2. Задание

Требуется выполнить следующие действия [1]:

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 одну из линейных моделей, SVM и дерево решений. Оцените качество модели с помощью трех подходящих для задачи метрик. Сравните качество полученных моделей.
5. Произведите для каждой модели подбор одного гиперпараметра с использованием `GridSearchCV` и кросс-валидации.
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 [2, 3]:

[1]:
```
from datetime import datetime
import graphviz
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import Lasso, LinearRegression
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
from sklearn.svm import NuSVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_graphviz, plot_tree

# 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]:

[2]:
```
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]:

[3]:
```
data = pd.read_csv("./SolarPrediction.csv")
```

Преобразуем временные колонки в соответствующий временной формат:

[4]:
```
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:

[5]:
```
data.dtypes
```

[5]:
```
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:

[6]:
```
data.head()
```

[6]:
```
      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. 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. Для преобразования времени в секунды используем следующий метод [6]:

[7]:
```
def time_to_second(t):
    return ((datetime.combine(datetime.min, t) - datetime.min)
            .total_seconds())
```

[8]:
```
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()
```

[8]:
```
   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
```

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

[9]:
```
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:

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

[10]:
```
(32686, 7)
```

Проверим основные статистические характеристики набора данных:

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

[11]:
```
          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:

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

[12]:
```
Radiation        0
Temperature      0
Pressure         0
Humidity         0
```

```
WindDirection    0
Speed            0
DayPart          0
dtype: int64
```

3.2. Разделение данных

Разделим данные на целевой столбец и признаки:

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

[14]:
```
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
```

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

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

Предобработаем данные, чтобы методы работали лучше:

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

[16]:
```
        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
```

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)
```

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

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. Линейная модель — Lasso

Попробуем метод Lasso с гиперпараметром $\alpha = 1$:

[20]:
```
las_1 = Lasso(alpha=1.0)
las_1.fit(X_train, y_train)
```

[20]:
```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Проверим метрики построенной модели:

[21]:
```
test_model(las_1)
```

```
mean_absolute_error: 156.39773885479397
median_absolute_error: 122.53656019076396
r2_score: 0.5959528719710016
```

Видно, что данный метод без настройки гиперпараметров несколько хуже, чем метод K ближайших соседей.

3.3.2. SVM

Попробуем метод NuSVR с гиперпараметром $\nu = 0{,}5$:

[22]:
```
nusvr_05 = NuSVR(nu=0.5, gamma='scale')
nusvr_05.fit(X_train, y_train)
```

[22]:
```
NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='scale',␣
↪kernel='rbf',
      max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)
```

Проверим метрики построенной модели:

[23]:
```
test_model(nusvr_05)
```

```
mean_absolute_error: 113.30399649196396
median_absolute_error: 52.28354239843286
r2_score: 0.677863113632347
```

Внезапно SVM показал результаты хуже по средней абсолютной ошибке и коэффициенте детерминации. Однако медианная абсолютная ошибка меньше, чем у метода Lasso.

3.3.3. Дерево решений

Попробуем дерево решений с неограниченной глубиной дерева:

[24]:
```
dt_none = DecisionTreeRegressor(max_depth=None)
dt_none.fit(X_train, y_train)
```

[24]:
```
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:

[25]:
```
test_model(dt_none)
```

```
mean_absolute_error: 49.95265540871267
median_absolute_error: 0.7250000000000012
r2_score: 0.8329923378031585
```

Дерево решений показало прямо-таки очень хороший результат по сравнению с рассмотренными раньше методами. Оценим структуру получившегося дерева решений:

[26]:
```
def stat_tree(estimator):
    n_nodes = estimator.tree_.node_count
    children_left = estimator.tree_.children_left
    children_right = estimator.tree_.children_right

    node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
    is_leaves = np.zeros(shape=n_nodes, dtype=bool)
    stack = [(0, -1)]  # seed is the root node id and its parent depth
    while len(stack) > 0:
        node_id, parent_depth = stack.pop()
        node_depth[node_id] = parent_depth + 1

        # If we have a test node
        if (children_left[node_id] != children_right[node_id]):
            stack.append((children_left[node_id], parent_depth + 1))
            stack.append((children_right[node_id], parent_depth + 1))
        else:
            is_leaves[node_id] = True

    print("Всего узлов:", n_nodes)
    print("Листовых узлов:", sum(is_leaves))
    print("Глубина дерева:", max(node_depth))
    print("Минимальная глубина листьев дерева:",␣
 ↪min(node_depth[is_leaves]))
    print("Средняя глубина листьев дерева:", node_depth[is_leaves].mean())
```

[27]:
```
stat_tree(dt_none)
```

```
Всего узлов: 42969
Листовых узлов: 21485
Глубина дерева: 43
Минимальная глубина листьев дерева: 7
Средняя глубина листьев дерева: 20.744845240865722
```

3.4. Подбор гиперпараметра K

3.4.1. Линейная модель — Lasso

Введем список настраиваемых параметров:

[28]:
```
param_range = np.arange(0.001, 2.01, 0.1)
tuned_parameters = [{'alpha': param_range}]
tuned_parameters
```

[28]:
```
[{'alpha': array([1.000e-03, 1.010e-01, 2.010e-01, 3.010e-01, 4.010e-01,
5.010e-01,
         6.010e-01, 7.010e-01, 8.010e-01, 9.010e-01, 1.001e+00, 1.101e+00,
         1.201e+00, 1.301e+00, 1.401e+00, 1.501e+00, 1.601e+00, 1.701e+00,
         1.801e+00, 1.901e+00, 2.001e+00])}]
```

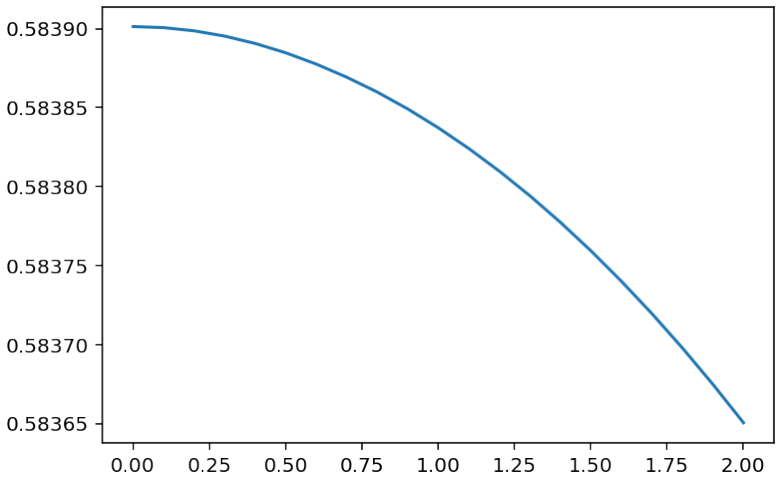
Запустим подбор параметра:

[29]:
```
gs = GridSearchCV(Lasso(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[29]:
```
Lasso(alpha=0.001, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

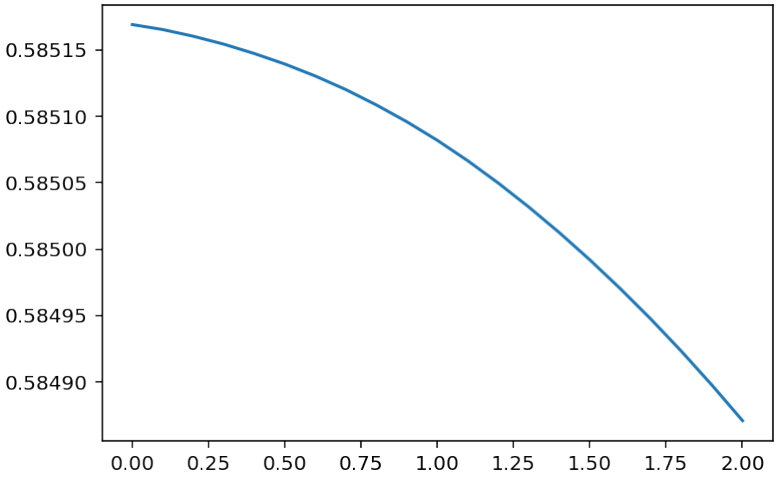
[30]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



Видно, что метод Lasso здесь не особо хорошо справляется, и здесь, скорее всего, было бы достаточно обычной линейной регрессии (в которую сходится Lasso при $\alpha = 0$).

На тестовом наборе данных картина ровно та же:

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



Будем считать, что GridSearch показал, что нам нужна обычная линейная регрессия:

[32]:
```
reg = LinearRegression()
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 156.41472692069644
median_absolute_error: 122.73509263147955
r2_score: 0.5961416061536914
```

В целом получили примерно тот же результат. Очевидно, что проблема в том, что данный метод не может дать хороший результат для данной выборки.

3.4.2. SVM

Введем список настраиваемых параметров:

[33]:
```
param_range = np.arange(0.1, 1.01, 0.1)
tuned_parameters = [{'nu': param_range}]
tuned_parameters
```

[33]:
```
[{'nu': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])}]
```

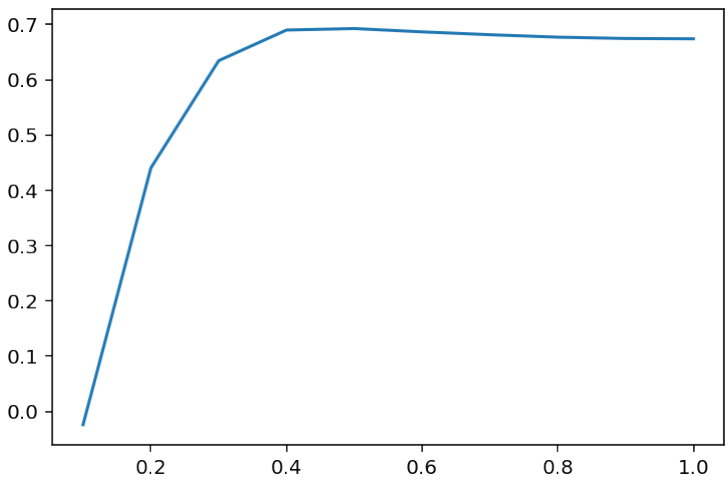
Запустим подбор параметра:

[34]:
```
gs = GridSearchCV(NuSVR(gamma='scale'), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[34]:
```
NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='scale',␣
↪kernel='rbf',
      max_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False)
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

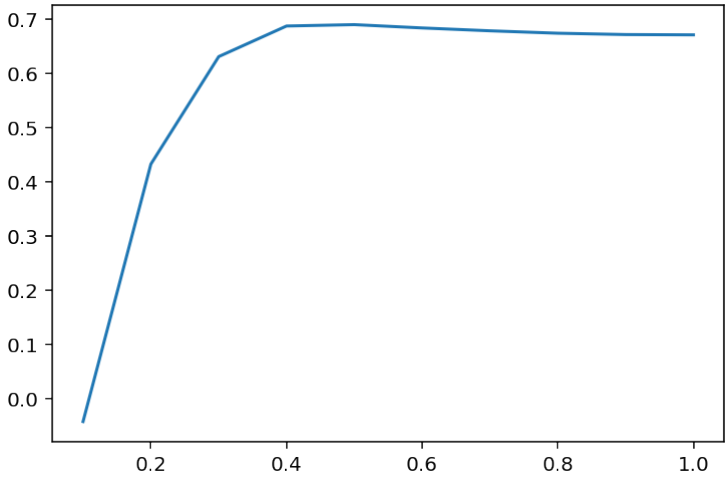
[35]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



Видно, что метод NuSVR справляется лучше, но не глобально. При этом также видно, что разработчики библиотеки scikit-learn провели хорошую работу: получившееся оптимальное значение $\nu = 0{,}5$ является стандартным для данного алгоритма [7].

На тестовом наборе данных картина ровно та же:

[36]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Так как параметры подобраны те же, то и обучение модели заново производить не будем.

3.4.3. Дерево решений

Введем список настраиваемых параметров:

[37]:
```
param_range = np.arange(1, 51, 2)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

[37]:
```
[{'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:

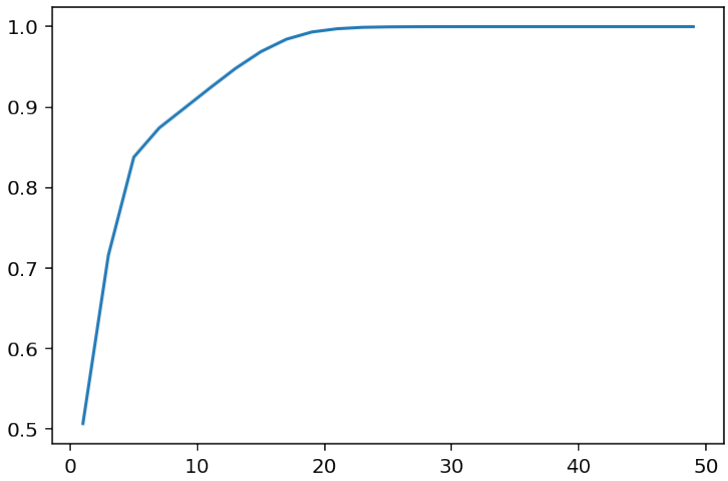
[38]:
```
gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[38]:
```
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:

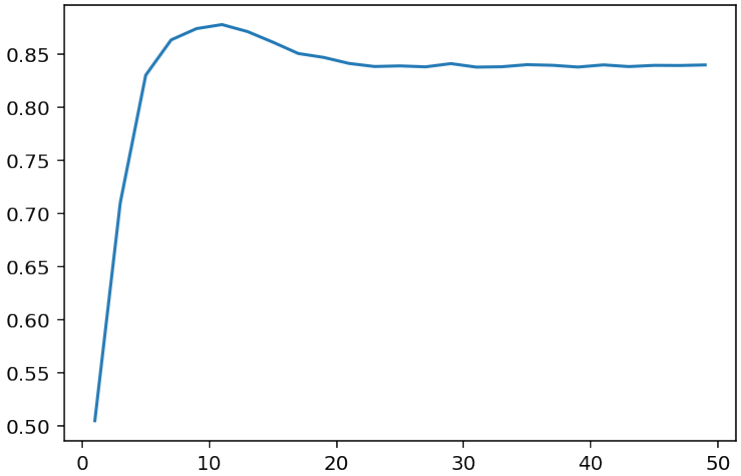
[39]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



Видно, что на тестовой выборке модель легко переобучается.

На тестовом наборе данных картина интереснее:

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



Проведем дополнительное исследование в районе пика.

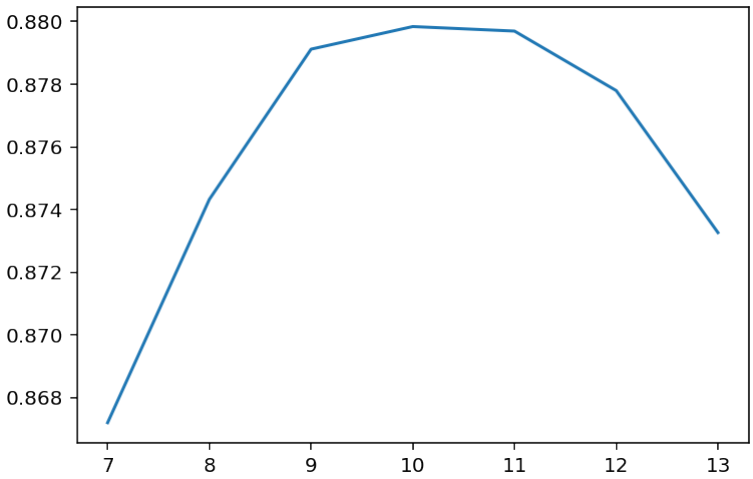
[41]:
```
param_range = np.arange(7, 14, 1)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

[41]:
```
[{'max_depth': array([ 7,  8,  9, 10, 11, 12, 13])}]
```

[42]:
```
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_
```

[42]:
```
DecisionTreeRegressor(criterion='mse', max_depth=10, 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')
```

[43]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Получили, что глубину дерева необходимо ограничить 10 уровнями. Проверим этот результат.

[44]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 49.37458067357269
median_absolute_error: 0.9516783975792638
r2_score: 0.8724040240942483
```

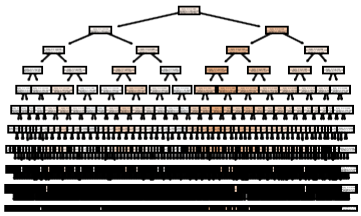
Вновь посмотрим статистику получившегося дерева решений.

[45]:
```
stat_tree(reg)
```

```
Всего узлов: 1711
Листовых узлов: 856
Глубина дерева: 10
Минимальная глубина листьев дерева: 7
Средняя глубина листьев дерева: 9.850467289719626
```

В целом получили примерно тот же результат. Коэффициент детерминации оказался немного выше, тогда как абсолютные ошибки также стали немного выше. Видно, что дерево решений достигло своего предела. При этом весьма поразительно, насколько хорошо данный метод решил задачу регрессии. Посмотрим на построенное дерево.

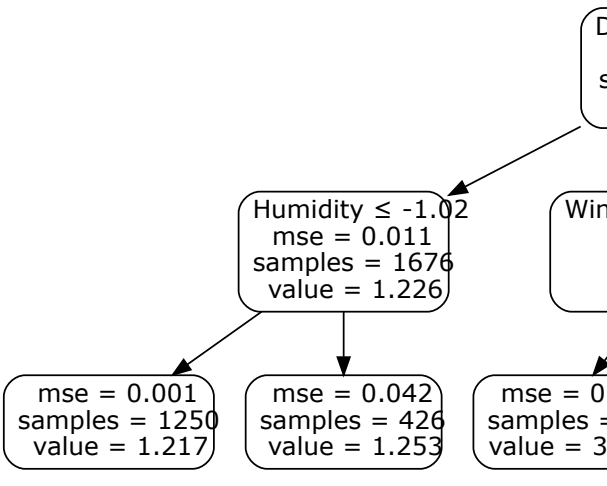
[46]:
```
plot_tree(reg, filled=True);
```



Вывод функции `plot_tree` выглядит весьма странно. Видимо, для настолько больших деревьев решений она не предназначена. Возможно, это со временем будет исправлено, так как эту функциональность только недавно добавили.

[47]:
```
dot_data = export_graphviz(reg, out_file=None, feature_names=columns,
                           filled=True, rounded=True,
                           special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

[47]:


Humidity ≤ -1.02
mse = 0.011
samples = 1676
value = 1.226
mse = 0.001
samples = 1250
value = 1.217
mse = 0.042
samples = 426
value = 1.253

Такое дерево уже можно анализировать. Видно, что сгенерировалось огромное множество различных условий, и, фактически, модель переобучена, но с другой стороны дерево решений и не могло быть построено иначе для задачи регрессии. К тому же на тестовой выборке данное дерево работает также довольно хорошо, так что, возможно, оно имеет право на существование. Если бы стояла задача классификации, то дерево решений явно показало бы себя просто отлично.

Список литературы

[1] Гапанюк Ю. Е. Лабораторная работа «Линейные модели, SVM и деревья решений» [Электронный ресурс] // GitHub. — 2019. — Режим доступа: `https://github.com/ugapanyuk/ml_course/wiki/LAB_TREES` (дата обращения: 19.04.2019).

[2] 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).

[3] Waskom M. seaborn 0.9.0 documentation [Electronic resource] // PyData. — 2018. — Access mode: `https://seaborn.pydata.org/` (online; accessed: 20.02.2019).

[4] pandas 0.24.1 documentation [Electronic resource] // PyData. — 2019. — Access mode: `http://pandas.pydata.org/pandas-docs/stable/` (online; accessed: 20.02.2019).

[5] dronio. Solar Radiation Prediction [Electronic resource] // Kaggle. — 2017. — Access mode: `https://www.kaggle.com/dronio/SolarEnergy` (online; accessed: 18.02.2019).

[6] Chrétien M. Convert datetime.time to seconds [Electronic resource] // Stack Overflow. — 2017. — Access mode: `https://stackoverflow.com/a/44823381` (online; accessed: 20.02.2019).

[7] scikit-learn 0.20.3 documentation [Electronic resource]. — 2019. — Access mode: `https://scikit-learn.org/` (online; accessed: 05.04.2019).