

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 ИУ5-21М
Хтет Мин Паинг Вин

Москва — 2020 г.

1. Цель лабораторной работы

Изучить линейные модели, SVM и деревья решений .

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

```
In [0]: 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
%matplotlib inline
# Set plots formats to save high resolution PNG
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

```
In [0]: pd.set_option("display.width", 70)
```

```
In [0]: data = pd.read_csv("./SolarPrediction.csv")
```

```
In [0]: 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 [0]: data.dtypes

Out[0]: UNIXTime int64
Data object
Time object
Radiation float64
Temperature int64
Pressure float64
Humidity int64
WindDirection float64
Speed float64
TimeSunRise object
TimeSunSet object
dtype: object

In [0]: data.head()

Out[0]:

	UNIXTime	Data	Time	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	TimeSunRise	Time
0	1475229326	9/29/2016 12:00:00 AM	23:55:26	1.21	48	30.46	59	177.39	5.62	06:13:00	18:13
1	1475229023	9/29/2016 12:00:00 AM	23:50:23	1.21	48	30.46	58	176.78	3.37	06:13:00	18:13
2	1475228726	9/29/2016 12:00:00 AM	23:45:26	1.23	48	30.46	57	158.75	3.37	06:13:00	18:13
3	1475228421	9/29/2016 12:00:00 AM	23:40:21	1.21	48	30.46	60	137.71	3.37	06:13:00	18:13
4	1475228124	9/29/2016 12:00:00 AM	23:35:24	1.17	48	30.46	62	104.95	5.62	06:13:00	18:13

```
In [0]: def time_to_second(t):  
        return ((datetime.combine(datetime.min, t) - datetime.min)  
                .total_seconds())
```

```
In [0]: 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[0]:
```

	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 [0]: df.dtypes
```

```
Out[0]: Radiation    float64  
        Temperature  int64  
        Pressure     float64  
        Humidity     int64  
        WindDirection float64  
        Speed        float64  
        DayPart      float64  
        dtype: object
```

```
In [0]: df.shape
```

```
Out[0]: (32686, 7)
```

In [0]: df.describe()

Out[0]:

	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

In [0]: df.isnull().sum()

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

In [0]: X = df.drop("Radiation", axis=1)
y = df["Radiation"]

```
In [0]: 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
```

```
In [0]: print(X.shape)
print(y.shape)
```

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

```
In [0]: columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

```
Out[0]:
```

	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	8.257741e-15	-8.589409e-14	9.563964e-16	-6.186353e-16	-2.072571e-14	-2.846377e-17
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


```
In [0]: X_train, X_test, y_train, y_test = train_test_split(X, y,  
test_size=0.25, random_state=346705925)
```

```
In [0]: print(X_train.shape)  
print(X_test.shape)  
print(y_train.shape)  
print(y_test.shape)
```

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

```
In [0]: 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)))
```

```
In [0]: las_1 = Lasso(alpha=1.0)  
las_1.fit(X_train, y_train)
```

```
Out[0]: 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)
```

```
In [0]: test_model(las_1)
```

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

```
In [0]: nusvr_05 = NuSVR(nu=0.5, gamma='scale')  
nusvr_05.fit(X_train, y_train)
```

```
Out[0]: 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)
```

In [0]: test_model(nusvr_05)

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

In [0]: dt_none = DecisionTreeRegressor(max_depth=None)
dt_none.fit(X_train, y_train)

Out[0]: 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')

In [0]: test_model(dt_none)

```
mean_absolute_error: 49.537692119432194
median_absolute_error: 0.7199999999999998
r2_score: 0.8324824754590788
```

In [0]: param_range = np.arange(0.001, 2.01, 0.1)
tuned_parameters = [{'alpha': param_range}]
tuned_parameters

Out[0]: [{'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])}]

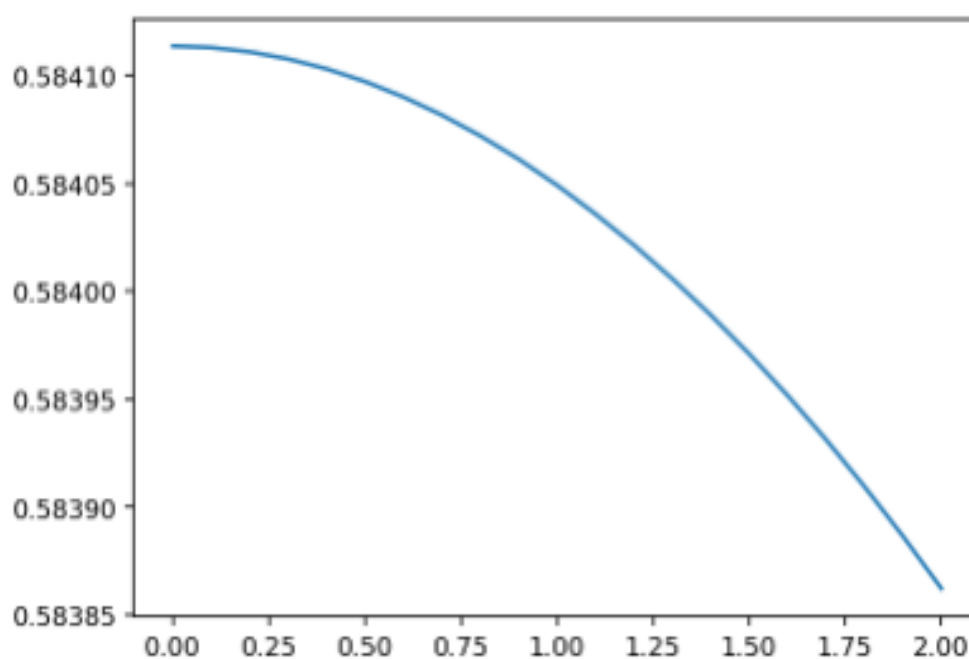
In [0]: gs = GridSearchCV(Lasso(), tuned_parameters,
cv=ShuffleSplit(n_splits=10), scoring="r2",
return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_

Out[0]: Lasso(alpha=0.101, 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)

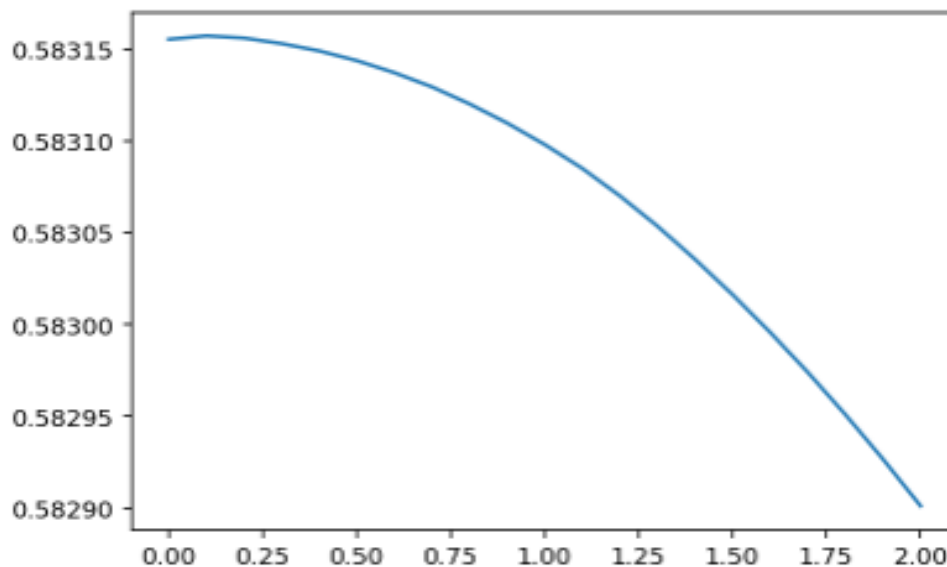
```
In [0]: gs = GridSearchCV(Lasso(), tuned_parameters,  
cv=ShuffleSplit(n_splits=10), scoring="r2",  
return_train_score=True, n_jobs=-1)  
gs.fit(X, y)  
gs.best_estimator_
```

```
Out[0]: Lasso(alpha=0.101, 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)
```

```
In [0]: plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



```
In [0]: plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



```
In [0]: reg = LinearRegression()  
reg.fit(X_train, y_train)  
test_model(reg)
```

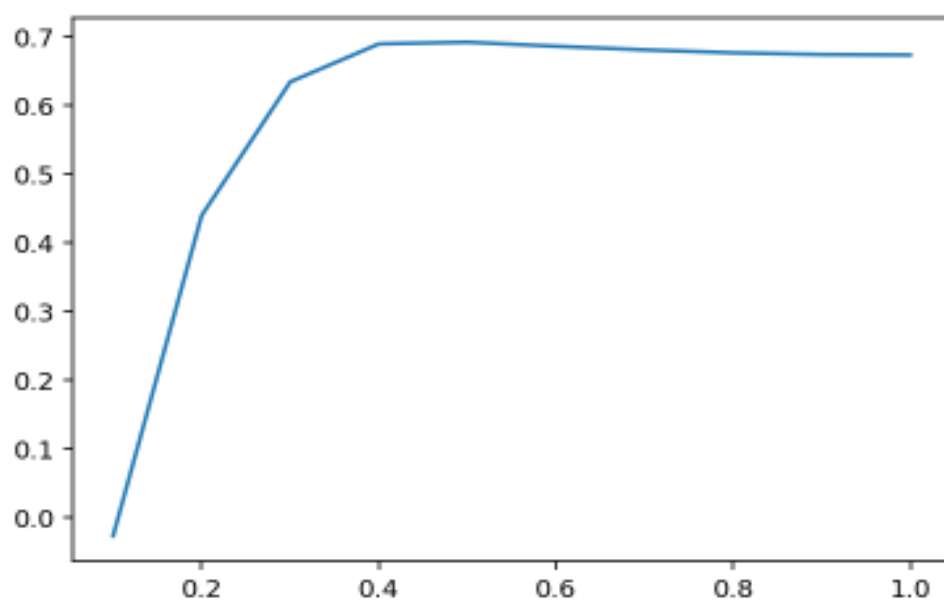
```
mean_absolute_error: 156.4147269206964  
median_absolute_error: 122.73509263147989  
r2_score: 0.5961416061536915
```

```
In [0]: param_range = np.arange(0.1, 1.01, 0.1)  
tuned_parameters = [{'nu': param_range}]  
tuned_parameters
```

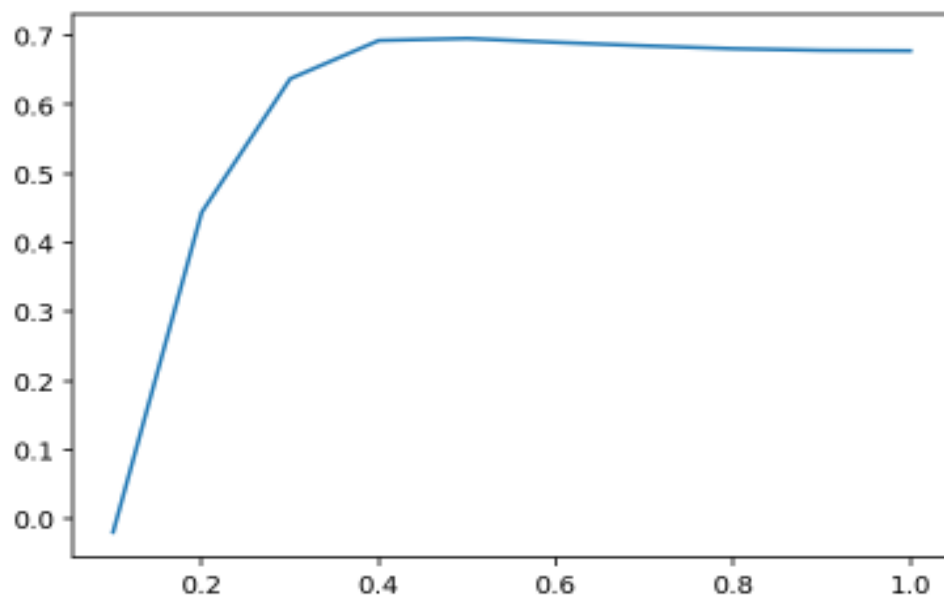
```
Out[0]: [{'nu': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])}]
```

```
In [0]: 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_
```

```
In [0]: plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



```
In [0]: plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```

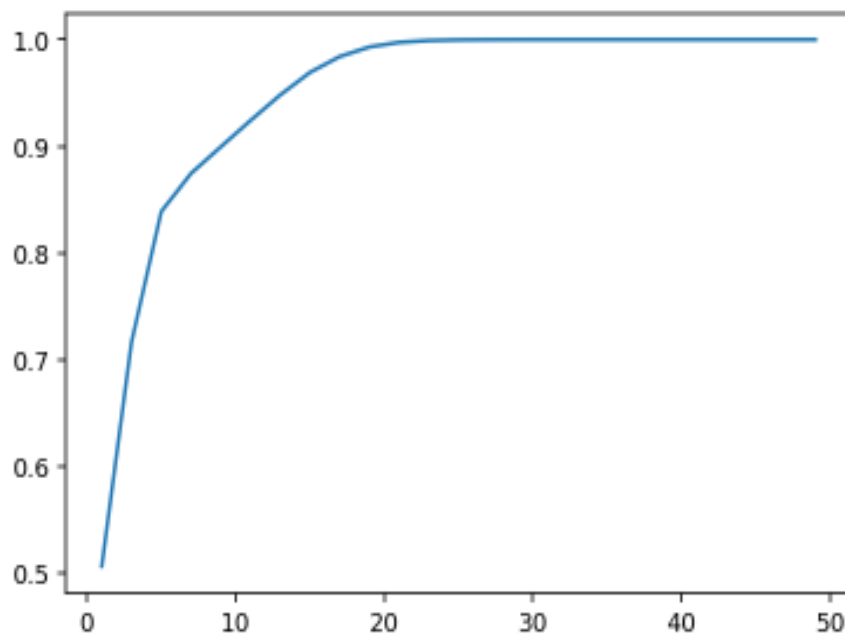


```
In [0]: param_range = np.arange(1, 51, 2)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

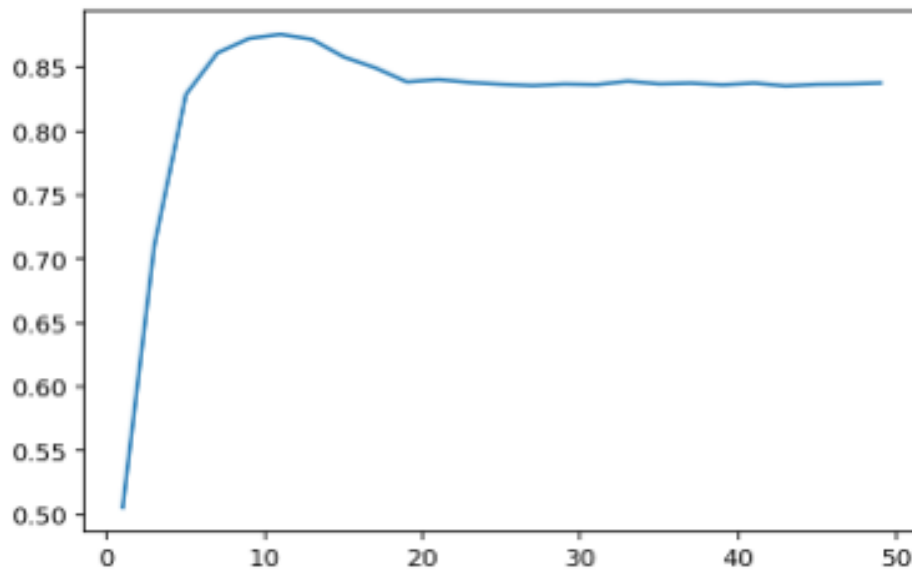
```
Out[0]: [{'max_depth': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
        35, 37, 39, 41, 43, 45, 47, 49])}]
```

```
In [0]: 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
```

```
In [89]: plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



In [90]: `plt.plot(param_range, gs.cv_results_["mean_test_score"]);`



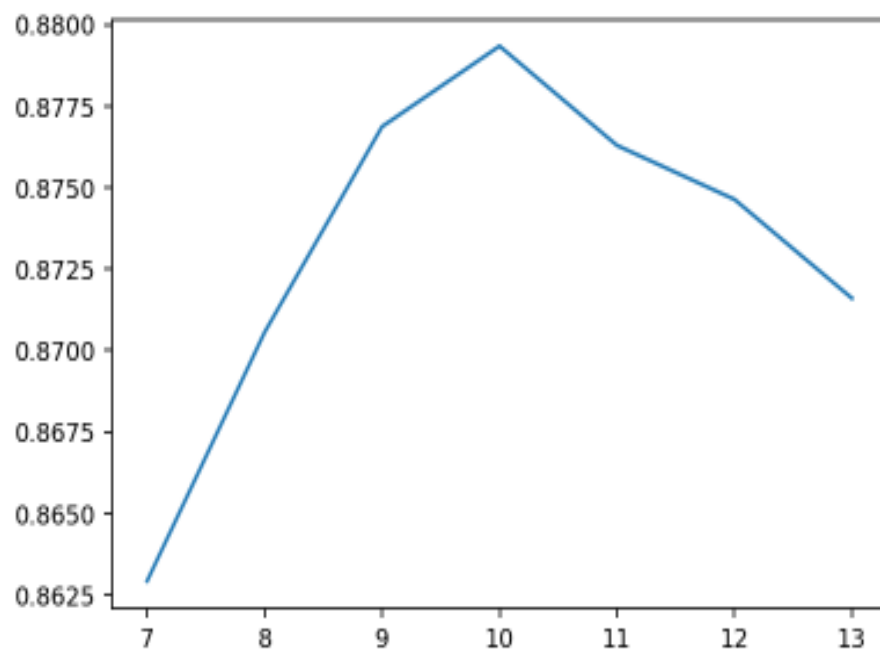
In [91]: `param_range = np.arange(7, 14, 1)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters`

Out[91]: `[{'max_depth': array([7, 8, 9, 10, 11, 12, 13])}]`

In [92]: `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_`

Out[92]: `DecisionTreeRegressor(ccp_alpha=0.0, 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='deprecated',
random_state=None, splitter='best')`

```
In [93]: plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



```
In [94]: reg = gs.best_estimator_  
reg.fit(X_train, y_train)  
test_model(reg)
```

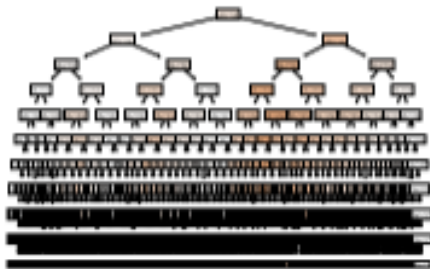
```
mean_absolute_error: 49.53670715948276  
median_absolute_error: 0.945844490216269  
r2_score: 0.870914100027162
```

```
In [0]: stat_tree(reg)
```


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.

```
In [0]: stat_tree(reg)
```

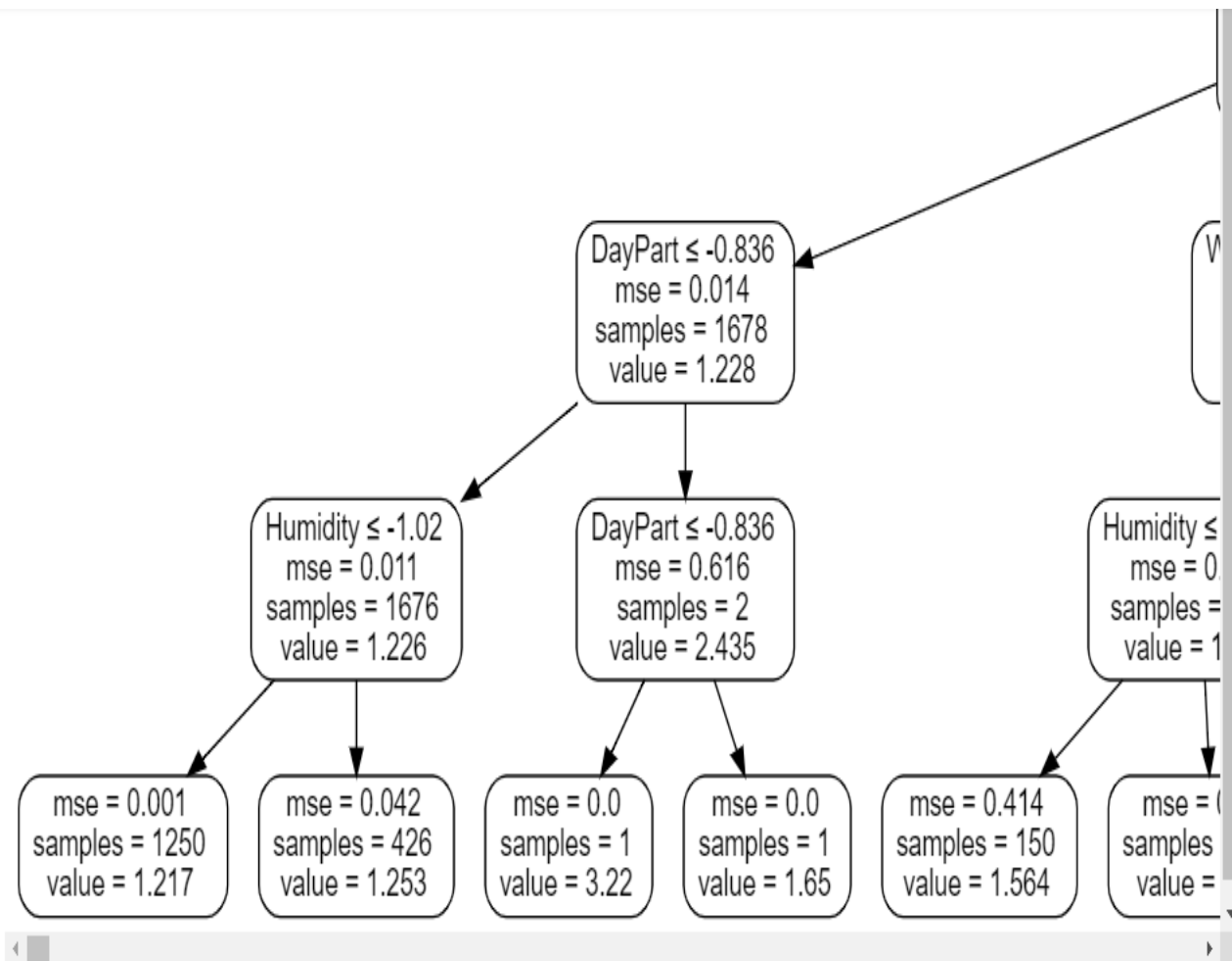
```
In [96]: plot_tree(reg, filled=True);
```



```
In [97]: dot_data = export_graphviz(reg, out_file=None, feature_names=columns,
    filled=True, rounded=True,
    special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

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.

[97]



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. — Accessmode: <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).

