

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

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

ОТЧЕТ

Лабораторная работа № 5
по дисциплине «Методы машинного обучения»

Тема: «Линейные модели, SVM и деревья решений.»

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 следующие модели:
 - одну из линейных моделей;
 - SVM;
 - дерево решений.
5. Оцените качество моделей с помощью трех подходящих для задачи метрик. Сравните качество полученных моделей.
6. Произведите для каждой модели подбор одного гиперпараметра с использованием `GridSearchCV` и кросс-валидации.
7. Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

Реализация задания

```
#подключаем данные
data = pd.read_csv('SolarPrediction.csv', sep=",")
```

```
# Список колонок с типами данных
data.dtypes
```

UNIXTime	int64
Data	object
Time	object
Radiation	float64
Temperature	int64
Pressure	float64
Humidity	int64
WindDirection(Degrees)	float64
Speed	float64
TimeSunRise	object
TimeSunSet	object
dtype:	object

```
data.head()
```

	UNIXTime	Data	Time	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	TimeSunRise	TimeSunSet
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	18:13:00
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	18:13:00
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	18:13:00
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	18:13:00
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	18:13:00

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 print(X.head(), "\n")
4 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 print(X.shape)
2 print(y.shape)
```

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

```
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.

Линейная модель — Lasso

```
1 #Попробуем метод Lasso с гиперпараметром alpha=1:  
2 las_1 = Lasso(alpha=1.0)  
3 las_1.fit(X_train, y_train)
```

```
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)
```

```
1 test_model(las_1)
```

```
mean_absolute_error: 157.96025861976267  
median_absolute_error: 124.33161677031507  
r2_score: 0.5910368441088809
```

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

SVM

```
1 #Попробуем метод NuSVR с гиперпараметром nu=0.5:  
2 nusvr_05 = NuSVR(nu=0.5, gamma='scale')  
3 nusvr_05.fit(X_train, y_train)
```

```
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)
```

```
1 test_model(nusvr_05)
```

```
mean_absolute_error: 172.92453188479877  
median_absolute_error: 101.9877834943342  
r2_score: 0.41677135378183905
```

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

Дерево решений

```
1 #Попробуем дерево решений с неограниченной глубиной дерева:  
2 dt_none = DecisionTreeRegressor(max_depth=None)  
3 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')
```

```
1 test_model(dt_none)
```

```
mean_absolute_error: 49.239755261869796  
median_absolute_error: 0.69  
r2_score: 0.8366668653991689
```

Дерево решений показало хороший результат по сравнению с рассмотренными раньше методами.

Оценим структуру получившегося дерева решений:

```
26 #Оценим структуру получившегося дерева решений:  
27 stat_tree(dt_none)
```

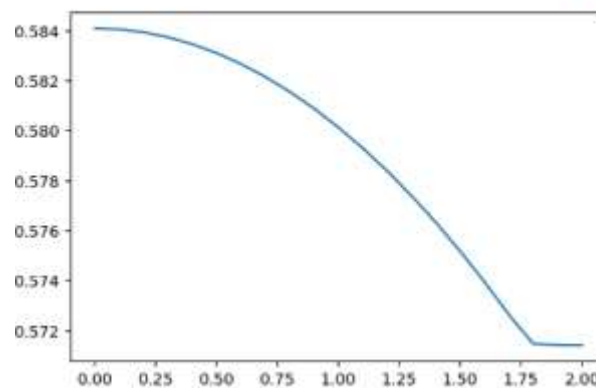
Всего узлов: 42969
Листовых узлов: 21485
Глубина дерева: 43
Минимальная глубина листьев дерева: 7
Средняя глубина листьев дерева: 20.743914358855015

Часть 4. Подбор гиперпараметра K

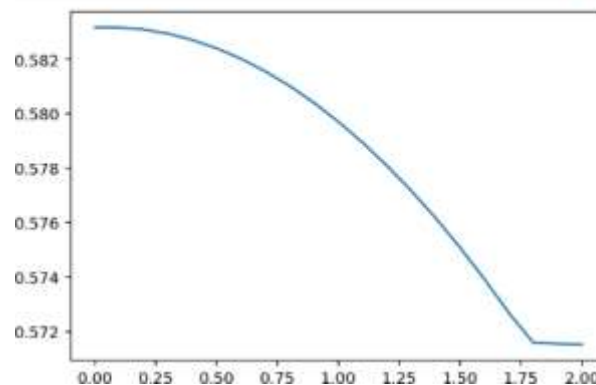
Линейная модель — Lasso

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

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

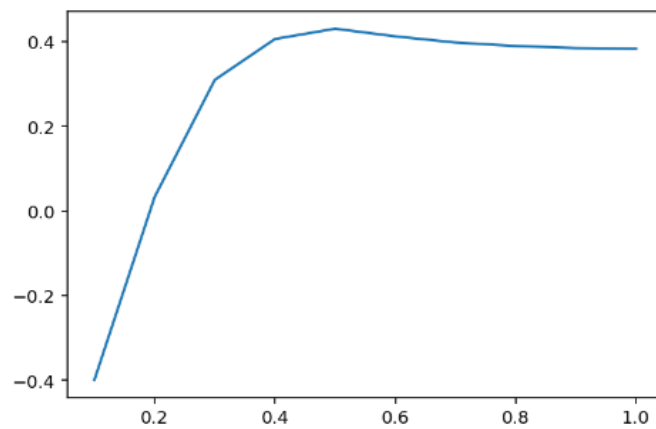
mean_absolute_error: 156.41472692069752
median_absolute_error: 122.7350926314856
r2_score: 0.5961416061536914

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

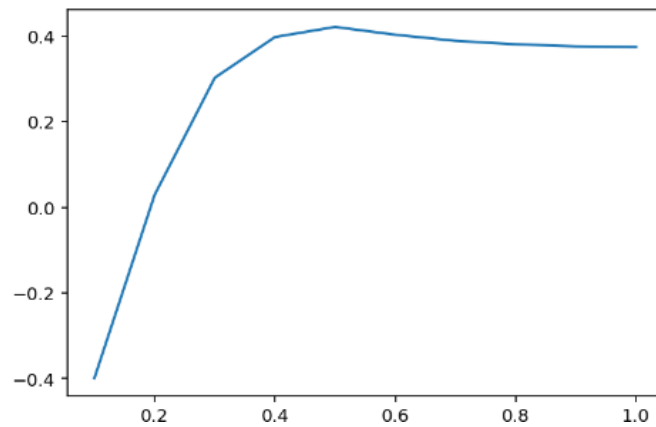
SVM

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"]);
```

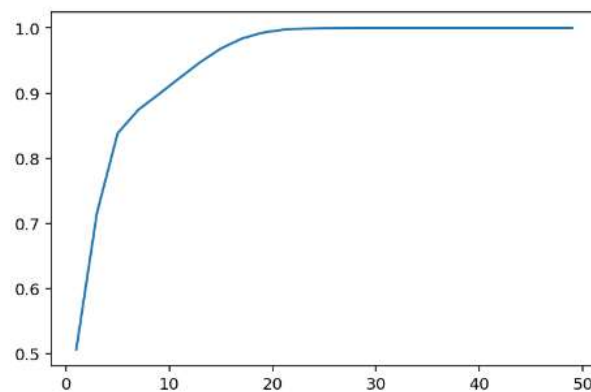


Видно, что метод NuSVR справляется лучше, но не глобально. При этом также видно, получившееся оптимальное значение $\nu=0,5$ является стандартным для данного алгоритма.

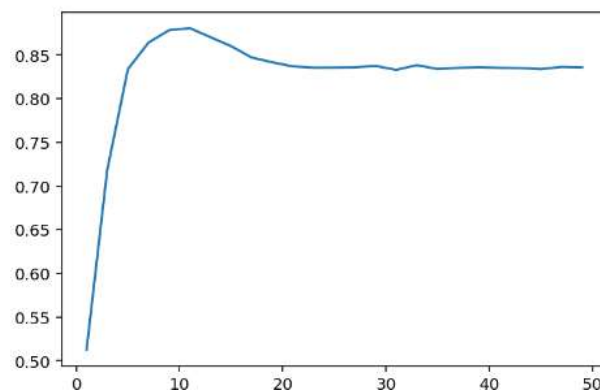
Дерево решений

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

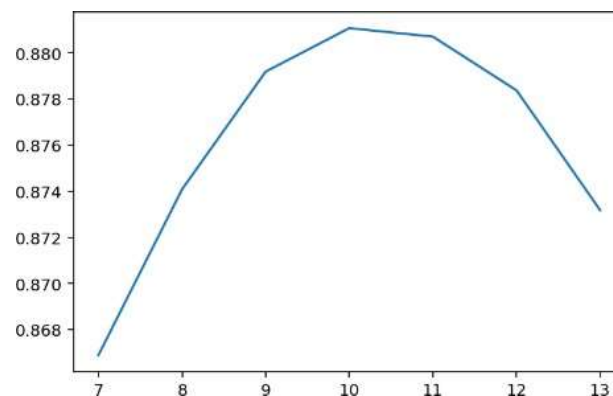
```
1 plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



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



Проведём дополнительное исследование в районе пика:



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

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

```
mean_absolute_error: 49.19982366267469  
median_absolute_error: 0.9458444902162735  
r2_score: 0.8729318611050234
```


Вновь посмотрим статистику получившегося дерева решений.

1	<code>stat_tree(reg)</code>
---	-----------------------------

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.