Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»



**Отчет**

**по лабораторной работе № 5**

**по курсу «Технологии машинного обучения»**

**«Линейные модели, SVM и деревья решений»**

**ИСПОЛНИТЕЛЬ:**

Мехмандаров Мурад

Группа ИУ5-63

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "\_\_"\_\_\_\_\_\_\_\_\_\_\_2020г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "\_\_"\_\_\_\_\_\_\_\_\_\_\_2020г.

Москва 2020

**Цель лабораторной работы**

Изучить линейные модели, SVM и деревья решений.

# Задание

Требуется выполнить следующие действия:

1. Выбрать набор данных (датасет) для решения задачи классификации или регресии.
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 двух подходящих для задачи метрик. Сравните качество полученных моделей.

# Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков:

In [1]:

pd

.

set\_option

(

"display.width"

,

70

)

**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 А4:

In [2]:

## 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):

In [3]:

data

**=**

pd

.

read\_csv

(

"./SolarPrediction.csv"

)

Преобразуем временные колонки в соответствующий временной формат:

In [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:

In [5]:

data

.

dtypes

Out[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

()

Out[6]:

**UNIXTime Data Time Radiation Temperature Pressure Humidity WindDirection**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1475229326 | 9/29/2016  12:00:00  AM | 23:55:26 | 1.21 | 48 | 30.46 | 59 | 177.39 |
| **1** | 1475229023 | 9/29/2016  12:00:00  AM | 23:50:23 | 1.21 | 48 | 30.46 | 58 | 176.78 |
| **2** | 1475228726 | 9/29/2016  12:00:00  AM | 23:45:26 | 1.23 | 48 | 30.46 | 57 | 158.75 |
| **3** | 1475228421 | 9/29/2016  12:00:00  AM | 23:40:21 | 1.21 | 48 | 30.46 | 60 | 137.71 |
| **4** | 1475228124 | 9/29/2016  12:00:00  AM | 23:35:24 | 1.17 | 48 | 30.46 | 62 | 104.95 |

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:

In [7]:

**def**

time\_to\_second

(

t

):

**return**

((

datetime

.

combine

(

datetime

.

min

,

t

)

**-**

datetime

.

min

)

.

total\_seconds

())

[8]:

Out[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

()

**Radiation Temperature Pressure Humidity WindDirection Speed DayPart**

1. 1.21 48 30.46 59 177.39 5.62 1.475602
2. 1.21 48 30.46 58 176.78 3.37 1.468588
3. 1.23 48 30.46 57 158.75 3.37 1.461713
4. 1.21 48 30.46 60 137.71 3.37 1.454653
5. 1.17 48 30.46 62 104.95 5.62 1.447778

In [9]:

df

.

dtypes

Out[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:

In [10]:

df

.

shape

Out[10]:

(32686, 7)

Проверим основные статистические характеристики набора данных: [11]:

df

.

describe

()

Out[11]:

**Radiation Temperature Pressure Humidity WindDirection Speed**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 32686.000000 | 32686.000000 | 32686.000000 | 32686.000000 | 32686.000000 | 32686.000000 | 32 |
| **mean** | 207.124697 | 51.103255 | 30.422879 | 75.016307 | 143.489821 | 6.243869 |  |
| **std** | 315.916387 | 6.201157 | 0.054673 | 25.990219 | 83.167500 | 3.490474 |  |
| **min** | 1.110000 | 34.000000 | 30.190000 | 8.000000 | 0.090000 | 0.000000 |  |
| **25%** | 1.230000 | 46.000000 | 30.400000 | 56.000000 | 82.227500 | 3.370000 |  |
| **50%** | 2.660000 | 50.000000 | 30.430000 | 85.000000 | 147.700000 | 5.620000 |  |
| **75%** | 354.235000 | 55.000000 | 30.460000 | 97.000000 | 179.310000 | 7.870000 |  |
| **max** | 1601.260000 | 71.000000 | 30.560000 | 103.000000 | 359.950000 | 40.500000 |  |

Проверим наличие пропусков в данных:

In [12]:

df

.

isnull

().

sum

()

Out[12]:

Radiation 0

Temperature 0

Pressure 0

Humidity 0

WindDirection 0

Speed 0 DayPart 0 dtype: int64

## Разделение данных

Разделим данные на целевой столбец и признаки:

In [13]:

X

**=**

df

.

drop

(

"Radiation"

,

axis

**=**

1

)

y

**=**

df

[

"Radiation"

]

[14]:

print

(

X

.

head

()

,

"\n"

)

print

(

y

.

head

())

Temperature Pressure Humidity WindDirection Speed DayPart

1. 48 30.46 59 177.39 5.62 1.475602
2. 48 30.46 58 176.78 3.37 1.468588
3. 48 30.46 57 158.75 3.37 1.461713
4. 48 30.46 60 137.71 3.37 1.454653
5. 48 30.46 62 104.95 5.62 1.447778

1. 1.21
2. 1.21
3. 1.23
4. 1.21
5. 1.17

Name: Radiation, dtype: float64 In [15]:

print

(

X

.

shape

)

print

(

y

.

shape

)

(32686, 6) (32686,)

Предобработаем данные, чтобы методы работали лучше:

In

[16]:

Out[16]:

columns

**=**

X

.

columns

scaler

**=**

StandardScaler

()

X

**=**

scaler

.

fit\_transform

(

X

)

pd

.

DataFrame

(

X

,

columns

**=**

columns

).

describe

()

**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** | 5.565041e-16 | 2.904952e-14 | 1.391260e-17 | 6.956302e-17 | -9.738822e-17 | 5.217226e-18 |
| **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 |
| **75%** | 6.283995e-01 | 6.789742e-01 | 8.458578e-01 | 4.307058e-01 | 4.658840e-01 | 8.682924e-01 |
| **max** | 3.208603e+00 | 2.508053e+00 | 1.076717e+00 | 2.602741e+00 | 9.814329e+00 | 1.797910e+00 |

Разделим выборку на тренировочную и тестовую:

In [17]:

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.25, random\_state**=**346705925)

[18]:

(24514, 6)

(8172

,

6)

print

(

X\_train

.

shape

)

print

(

X\_test

.

shape

)

print

(

y\_train

.

shape

)

print

(

y\_test

.

shape

)

(24514,) (8172,)

## Обучение моделей

Напишем функцию, которая считает метрики построенной модели:

In [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))) **Линейная модель — Lasso**

Попробуем метод Lasso с гиперпараметром 𝛼 = 1:

In [20]:

las\_1

**=**

Lasso

(

alpha

**=**

1.0

)

las\_1

.

fit

(

X\_train

,

y\_train

)

Out[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:

In [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.

### SVM

Попробуем метод NuSVR с гиперпараметром 𝜈 = 0,5:

In [22]:

nusvr\_05

**=**

NuSVR

(

nu

**=**

0.5

,

gamma

**=**

'scale'

)

nusvr\_05

.

fit

(

X\_train

,

y\_train

)

Out[22]:

NuSVR(C=1.0, cache\_size=200, coef0=0.0, degree=3, gamma='scale', kerne l='rbf', max\_iter=-1, nu=0.5, shrinking=True, tol=0.001, verbose=False) Проверим метрики построенной модели:

In [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.

### Дерево решений

Попробуем дерево решений с неограниченной глубиной дерева:

In [24]:

dt\_none

**=**

DecisionTreeRegressor

(

max\_depth

**=**

**None**

)

dt\_none

.

fit

(

X\_train

,

y\_train

)

Out[24]:

DecisionTreeRegressor(criterion='mse', max\_depth=None, max\_features=No ne,

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='bes

t')

Проверим метрики построенной модели:

In [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:

In [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()) In [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