**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)

О Т Ч Е Т

**по лабораторной работе**

по дисциплине: Технологии машинного обучения

на тему: Линейные модели, SVM и деревья решений.

\_

\_

\_

\_

\_

\_

\_

\_

Студент РТ5-61Б **А.C. Акушко**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель

**Ю.Е. Гапанюк**

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

# Лабораторная работа №5

**Линейные модели, SVM и деревья решений**

## Цель лабораторной работы

Изучение линейных моделей, SVM и деревьев решений.

## 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.

1. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

## Дополнительные задания

Проведите эксперименты с важностью признаков в дереве решений. Визуализируйте дерево решений.

# Ход выполнения лабораторной работы

In [205]:

**import pandas as pd import seaborn as sns import numpy as np**

**from typing import** Tuple, Dict

**import matplotlib.pyplot as plt from operator import** itemgetter

**from sklearn.model\_selection import** train\_test\_split **from sklearn.linear\_model import** LinearRegression **from sklearn.preprocessing import** StandardScaler

**from sklearn.metrics import** f1\_score, r2\_score, mean\_squared\_error, mean\_absolute\_error, accur acy\_score, precision\_score

**from sklearn.svm import** LinearSVR, SVR

**from sklearn.tree import** DecisionTreeRegressor, plot\_tree

%**matplotlib** inline sns.set(style="ticks")

In [206]:

col\_list = ['Pelvic\_incidence',

'Pelvic\_tilt', 'Lumbar\_lordosis\_angle', 'Sacral\_slope', 'Pelvic\_radius', 'Degree\_spondylolisthesis', 'Pelvic\_slope',

'Direct\_tilt', 'Thoracic\_slope', 'Cervical\_tilt', 'Sacrum\_angle', 'Scoliosis\_slope', 'Class\_att', 'To\_drop']

data = pd.read\_csv('data/Dataset\_spine.csv', names=col\_list, header=1, sep=",") data.drop('To\_drop', axis=1, inplace=**True**)

In [207]:

data.head()

Out[207]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Pelvic\_incidence** | **Pelvic\_tilt** | **Lumbar\_lordosis\_angle** | **Sacral\_slope** | **Pelvic\_radius** | **Degree\_spondy** |
| **0** 39.056951 | 10.060991 | 25.015378 | 28.995960 | 114.405425 |  |
| **1** 68.832021 | 22.218482 | 50.092194 | 46.613539 | 105.985135 |  |
| **2** 69.297008 | 24.652878 | 44.311238 | 44.644130 | 101.868495 |  |
| **3** 49.712859 | 9.652075 | 28.317406 | 40.060784 | 108.168725 |  |
| **4** 40.250200 | 13.921907 | 25.124950 | 26.328293 | 130.327871 |  |

In [208]:

data.isnull().sum()

Out[208]: Pelvic\_incidence 0

Pelvic\_tilt 0

Lumbar\_lordosis\_angle 0

Sacral\_slope 0

Pelvic\_radius 0

Degree\_spondylolisthesis 0

|  |  |  |
| --- | --- | --- |
| Pelvic\_slope |  | 0 |
| Direct\_tilt | 0 |  |
| Thoracic\_slope |  | 0 |
| Cervical\_tilt | 0 |  |
| Sacrum\_angle |  | 0 |
| Scoliosis\_slope |  | 0 |
| Class\_att | 0 |  |
| dtype: int64 |  |  |

Пропуски данных отсутствуют.

In [209]:

data['Class\_att'] = data['Class\_att'].map({'Abnormal': 1, 'Normal': 0})

Разделим выборку на обучающую и тестовую:

In [210]:

*# Разделим данные на целевой столбец и признаки*

X = data.drop("Class\_att", axis=1) Y = data["Class\_att"]

In [211]:

*# С использованием метода train\_test\_split разделим выборку на обучающую и тестовую* X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=1) X\_train.shape, X\_test.shape, Y\_train.shape, Y\_test.shape

Out[211]: ((231, 12), (78, 12), (231,), (78,))

## Линейная модель

In [212]:

*#Построим корреляционную матрицу*

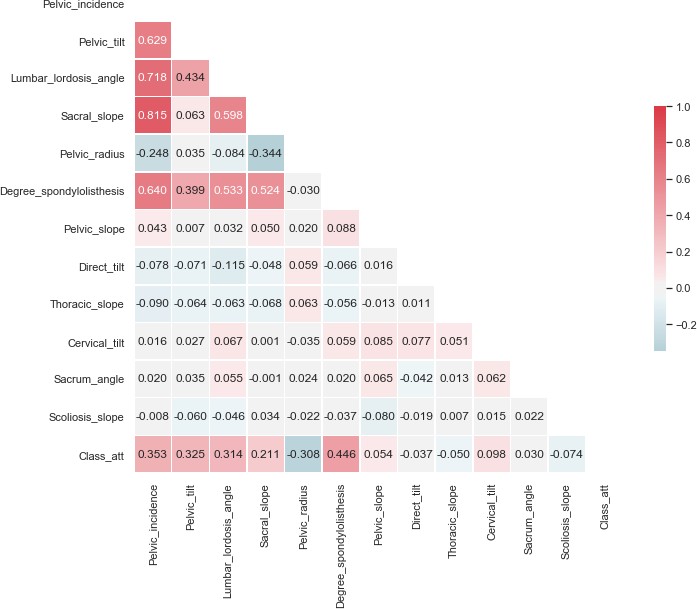
sns.set(style="white")

corr = data.corr(method='pearson')

mask = np.zeros\_like(corr, dtype=np.bool) mask[np.triu\_indices\_from(mask)] = **True** f, ax = plt.subplots(figsize=(11, 9))

cmap = sns.diverging\_palette(220, 10, as\_cmap=**True**)

g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=**True**, fmt='.3f', square=**True**, linewidths=.5, cbar\_kws={"shrink": .5})

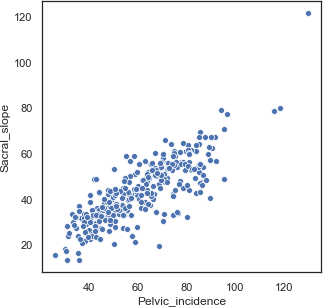


In [213]:

fig, ax = plt.subplots(figsize=(5,5))

sns.scatterplot(ax=ax, x='Pelvic\_incidence', y='Sacral\_slope', data=data)

Out[213]: <matplotlib.axes.\_subplots.AxesSubplot at 0x5d88a30>



In [214]:

x\_array = data['Pelvic\_incidence'].values y\_array = data['Sacral\_slope'].values

In [215]:

*# Аналитическое вычисление коэффициентов регрессии*

**def** analytic\_regr\_coef(x\_array : np.ndarray,

y\_array : np.ndarray) -> Tuple[float, float]: x\_mean = np.mean(x\_array)

y\_mean = np.mean(y\_array)

var1 = np.sum([(x-x\_mean)\*\*2 **for** x **in** x\_array])

cov1 = np.sum([(x-x\_mean)\*(y-x\_mean) **for** x, y **in** zip(x\_array, y\_array)]) b1 = cov1 / var1

b0 = y\_mean - b1\*x\_mean

**return** b0, b1

In [216]:

b0, b1 = analytic\_regr\_coef(x\_array, y\_array) b0, b1

Out[216]: (4.565546113493063, 0.6347707526286969)

In [217]:

*# Вычисление значений y на основе x для регрессии*

**def** y\_regr(x\_array : np.ndarray, b0: float, b1: float) -> np.ndarray: res = [b1\*x+b0 **for** x **in** x\_array]

**return** res

In [218]:

y\_array\_regr = y\_regr(x\_array, b0, b1)

In [219]:

*# Простейшая реализация градиентного спуска*

**def** gradient\_descent(x\_array : np.ndarray, y\_array : np.ndarray,

b0\_0 : float, b1\_0 : float, epochs : int,

learning\_rate : float = 0.001

) -> Tuple[float, float]:

*# Значения для коэффициентов по умолчанию*

b0, b1 = b0\_0, b1\_0

k = float(len(x\_array))

**for** i **in** range(epochs):

*# Вычисление новых предсказанных значений*

*# используется векторизованное умножение и сложение для вектора и константы*

y\_pred = b1 \* x\_array + b0

*# Расчет градиентов*

*# np.multiply - поэлементное умножение векторов*

dL\_db1 = (-2/k) \* np.sum(np.multiply(x\_array, (y\_array - y\_pred))) dL\_db0 = (-2/k) \* np.sum(y\_array - y\_pred)

*# Изменение значений коэффициентов:*

b1 = b1 - learning\_rate \* dL\_db1 b0 = b0 - learning\_rate \* dL\_db0

*# Результирующие значения* y\_pred = b1 \* x\_array + b0 **return** b0, b1, y\_pred

In [220]:

**def** show\_gradient\_descent(epochs, b0\_0, b1\_0):

grad\_b0, grad\_b1, grad\_y\_pred = gradient\_descent(x\_array, y\_array, b0\_0, b1\_0, epochs) print('b0 = **{}** - (теоретический), **{}** - (градиентный спуск)'.format(b0, grad\_b0)) print('b1 = **{}** - (теоретический), **{}** - (градиентный спуск)'.format(b1, grad\_b1)) print('MSE = **{}**'.format(mean\_squared\_error(y\_array\_regr, grad\_y\_pred))) plt.plot(x\_array, y\_array, 'g.')

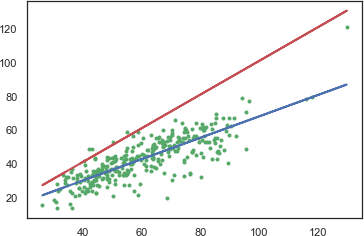
plt.plot(x\_array, y\_array\_regr, 'b', linewidth=2.0) plt.plot(x\_array, grad\_y\_pred, 'r', linewidth=2.0) plt.show()

In [221]:

*# Примеры использования градиентного спуска*

show\_gradient\_descent(0, 1, 1)

b0 = 4.565546113493063 - (теоретический), 1 - (градиентный спуск) b1 = 0.6347707526286969 - (теоретический), 1 - (градиентный спуск) MSE = 382.8630387134672

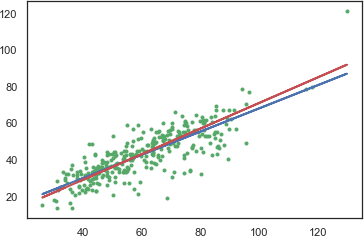


In [222]:

show\_gradient\_descent(0, 1, 0.7)

b0 = 4.565546113493063 - (теоретический), 1 - (градиентный спуск)

b1 = 0.6347707526286969 - (теоретический), 0.7 - (градиентный спуск) MSE = 1.4084724961222816



In [223]:

*# Обучим линейную регрессию и сравним коэффициенты с рассчитанными ранее*

reg1 = LinearRegression().fit(x\_array.reshape(-1, 1), y\_array.reshape(-1, 1)) (b1, reg1.coef\_), (b0, reg1.intercept\_)

Out[223]: ((0.6347707526286969, array([[0.63477075]])),

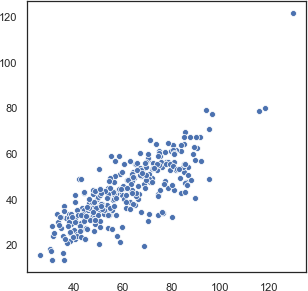
(4.565546113493063, array([4.56554611])))

## SVM

In [224]:

fig, ax = plt.subplots(figsize=(5,5)) sns.scatterplot(ax=ax, x=x\_array, y=y\_array)

Out[224]: <matplotlib.axes.\_subplots.AxesSubplot at 0xf20d730>



In [225]:

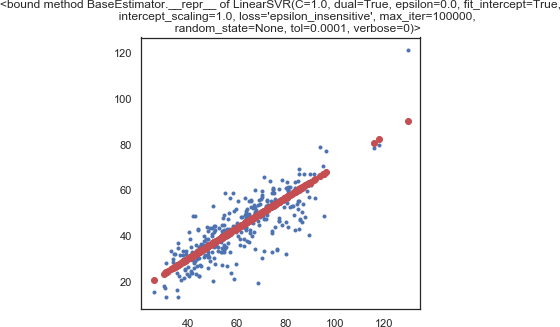
**def** plot\_regr(clf): title = clf. repr

clf.fit(x\_array.reshape(-1, 1), y\_array) y\_pred = clf.predict(x\_array.reshape(-1, 1)) fig, ax = plt.subplots(figsize=(5,5)) ax.set\_title(title)

ax.plot(x\_array, y\_array, 'b.') ax.plot(x\_array, y\_pred, 'ro') plt.show()

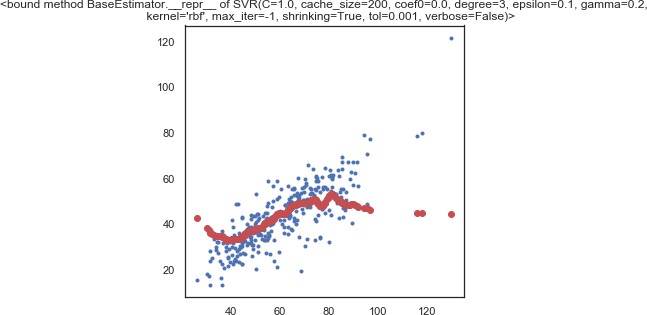
In [226]:

plot\_regr(LinearSVR(C=1.0, max\_iter=100000))



In [227]:

plot\_regr(SVR(kernel='rbf', gamma=0.2, C=1.0))



## Дерево решений

In [228]:

*# Обучим дерево на всех признаках*

tree = DecisionTreeRegressor(random\_state=1) tree.fit(X\_train, Y\_train)

Out[228]: 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=1, splitter='best')

In [229]:

*# Важность признаков*

list(zip(X\_train.columns.values, tree.feature\_importances\_))

Out[229]: [('Pelvic\_incidence', 0.03881985535831688),

('Pelvic\_tilt', 0.0648640652612829),

('Lumbar\_lordosis\_angle', 0.06149200087661627),

('Sacral\_slope', 0.08263904265020883),

('Pelvic\_radius', 0.181238439623055),

('Degree\_spondylolisthesis', 0.41411903317709764),

('Pelvic\_slope', 0.023065828402366866),

('Direct\_tilt', 0.01842735042735041),

('Thoracic\_slope', 0.07000532439464523),

('Cervical\_tilt', 0.007145299145299295),

('Sacrum\_angle', 0.038183760683760684),

('Scoliosis\_slope', 0.0)]

In [230]:

*# Важность признаков в сумме дает единицу*

sum(tree.feature\_importances\_)

Out[230]: 1.0

In [231]:

**def** draw\_feature\_importances(tree\_model, X\_dataset, figsize=(10,5)):

*"""*

*Вывод важности признаков в виде графика*

*"""*

*# Сортировка значений важности признаков по убыванию*

list\_to\_sort = list(zip(X\_dataset.columns.values, tree\_model.feature\_importances\_)) sorted\_list = sorted(list\_to\_sort, key=itemgetter(1), reverse = **True**)

*# Названия признаков*

labels = [x **for** x,\_ **in** sorted\_list]

*# Важности признаков*

data = [x **for** \_,x **in** sorted\_list]

*# Вывод графика*

fig, ax = plt.subplots(figsize=figsize) ind = np.arange(len(labels)) plt.bar(ind, data)

plt.xticks(ind, labels, rotation='vertical')

*# Вывод значений*

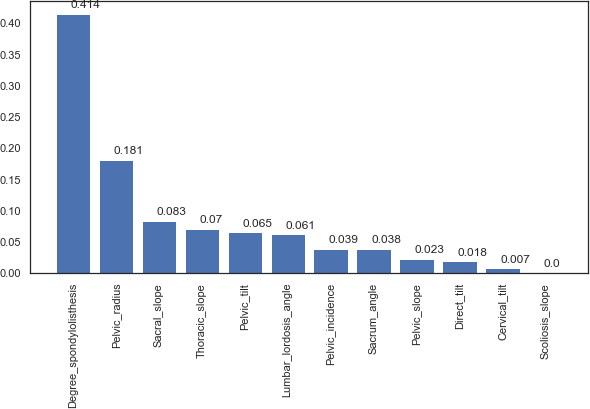
**for** a,b **in** zip(ind, data):

plt.text(a-0.05, b+0.01, str(round(b,3))) plt.show()

**return** labels, data

In [232]:

tree\_fl, tree\_fd = draw\_feature\_importances(tree, X\_train)



In [233]:

*# Список признаков, отсортированный на основе важности, и значения важности*

tree\_fl, tree\_fd

Out[233]: (['Degree\_spondylolisthesis',

'Pelvic\_radius', 'Sacral\_slope', 'Thoracic\_slope', 'Pelvic\_tilt', 'Lumbar\_lordosis\_angle', 'Pelvic\_incidence', 'Sacrum\_angle', 'Pelvic\_slope', 'Direct\_tilt', 'Cervical\_tilt', 'Scoliosis\_slope'], [0.41411903317709764,

0.181238439623055,

0.08263904265020883,

0.07000532439464523,

0.0648640652612829,

0.06149200087661627,

0.03881985535831688,

0.038183760683760684,

0.023065828402366866,

0.01842735042735041,

0.007145299145299295,

0.0])

In [234]:

X\_train.head()

Out[234]:

**Pelvic\_incidence Pelvic\_tilt Lumbar\_lordosis\_angle Sacral\_slope Pelvic\_radius Degree\_spon**

**291** 34.756738 2.631740 29.504381 32.124998 127.139850

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **17** | 38.697912 | 13.444749 | 31.000000 | 25.253163 | 123.159251 |
| **110** | 84.998956 | 29.610098 | 83.352194 | 55.388858 | 126.912990 |
| **228** | 43.436451 | 10.095743 | 36.032224 | 33.340707 | 137.439694 |
| **125** | 70.676898 | 21.704402 | 59.181161 | 48.972496 | 103.008355 |

In [235]:

*# Пересортируем признаки на основе важности* X\_train\_sorted = X\_train[tree\_fl] X\_train\_sorted.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Out[235]: |  | | | | | | |
|  |  | **Degree\_spondylolisthesis** | **Pelvic\_radius** | **Sacral\_slope** | **Thoracic\_slope** | **Pelvic\_tilt** | **Lumbar\_lor** |
|  | **291** | -0.460894 | 127.139850 | 32.124998 | 11.2762 | 2.631740 |  |
|  | **17** | 1.429186 | 123.159251 | 25.253163 | 17.9575 | 13.444749 |  |
|  | **110** | 71.321175 | 126.912990 | 55.388858 | 9.0119 | 29.610098 |  |
|  | **228** | -3.114451 | 137.439694 | 33.340707 | 11.0132 | 10.095743 |  |
|  | **125** | 27.810148 | 103.008355 | 48.972496 | 14.8568 | 21.704402 |  |

In [236]:

Y\_test\_predict = tree.predict(X\_test)

In [237]:

mean\_absolute\_error(Y\_test, Y\_test\_predict)

Out[237]: 0.11538461538461539

In [238]:

*# Обучим дерево и предскажем результаты на пяти лучших признаках*

tree\_2 = DecisionTreeRegressor(random\_state=1).fit(X\_train[tree\_fl[0:5]], Y\_train) Y\_test\_predict\_2 = tree\_2.predict(X\_test[tree\_fl[0:5]])

In [239]:

mean\_absolute\_error(Y\_test, Y\_test\_predict\_2)

Out[239]: 0.16666666666666666

In [240]:

*# Исследуем, как изменяется ошибка при добавлении признаков в порядке значимости*

X\_range = list(range(1, len(X\_train.columns)+1)) X\_range

Out[240]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

In [241]:

mae\_list = []

**for** i **in** X\_range:

*# Обучим дерево и предскажем результаты на заданном количестве признаков* tree\_3 = DecisionTreeRegressor(random\_state=1).fit(X\_train[tree\_fl[0:i]], Y\_train) Y\_test\_predict\_3 = tree\_3.predict(X\_test[tree\_fl[0:i]])

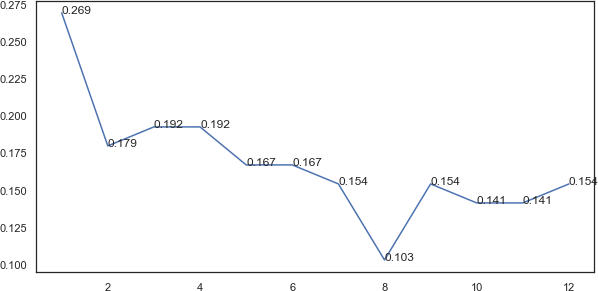
temp\_mae = mean\_absolute\_error(Y\_test, Y\_test\_predict\_3) mae\_list.append(temp\_mae)

In [242]:

plt.subplots(figsize=(10,5)) plt.plot(X\_range, mae\_list)

**for** a,b **in** zip(X\_range, mae\_list): plt.text(a, b, str(round(b,3)))

plt.show()



## Оценка качества моделей

Дерево решений

In [243]:

print("r2\_score:", r2\_score(Y\_test, tree.predict(X\_test))) print("mean\_squared\_error:", mean\_squared\_error(Y\_test, tree.predict(X\_test)))

r2\_score: 0.4701886792452832

mean\_squared\_error: 0.11538461538461539

Линейная регрессия

In [244]:

pred = reg1.predict(x\_array.reshape(-1, 1)) print("r2\_score:", r2\_score(y\_array, pred))

print("mean\_squared\_error", mean\_squared\_error(y\_array, pred))

r2\_score: 0.664423352506976

mean\_squared\_error 60.45739674813066

Метод опорных векторов

In [245]:

svr = SVR(kernel='rbf') svr.fit(X\_train, Y\_train)

print("r2\_score:", r2\_score(Y\_test, svr.predict(X\_test))) print("mean\_squared\_error", mean\_squared\_error(Y\_test, svr.predict(X\_test)))

r2\_score: 0.5993410854165501

mean\_squared\_error 0.08725724224573822

Последние две модели являются приемлемыми, т.к. коэффициент детерминации для всех трех моделей больше 50%.

Если учитывать показатели обеих метрик, наилучший результат показал метод опорных векторов.