

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизация производственных процессов»

Введение в Искусственный интеллект, Data science и  
обработку больших данных

Методические указания  
с заданиями по контрольной работе  
для студентов заочной формы обучения

Ростов-на-Дону  
ДГТУ  
2024

УДК 681.5

Составитель:

Методические указания. – Ростов-на-Дону : Донской гос. техн. ун-т, 2024. – 34 с.

Методические указания с заданиями по контрольной работе по дисциплине «Введение в Искусственный интеллект, Data science и обработку больших данных» предназначены для студентов заочной формы обучения по направлению подготовки 15.04.04 «Автоматизация технологических процессов и производств» профиль «Интеллектуальные системы сбора и анализа больших данных».

УДК 681.5

Печатается по решению редакционно-издательского совета  
Донского государственного технического университета

---

В печать \_\_\_\_ . \_\_\_\_ . 20 \_\_\_\_ г.  
Формат 60х84/16. Объем \_\_\_\_ усл. п. л.  
Тираж \_\_\_\_ экз. Заказ № \_\_\_\_ .

---

Издательский центр ДГТУ  
Адрес университета и полиграфического предприятия:  
344000, г. Ростов-на-Дону, пл. Гагарина, 1

© Донской государственный  
технический университет, 2024

## Содержание

ОБЩИЕ СВЕДЕНИЯ И ВЫБОР ВАРИАНТА .....	4
ЗАДАНИЯ ПЕРВОЙ ГРУППЫ .....	4
ЗАДАНИЯ ВТОРОЙ И ТРЕТЬЕЙ ГРУППЫ .....	6
ЗАДАНИЕ № 1.....	6
Алгоритм кластеризация методом $K$ средних.....	6
с применением евклидова расстояния .....	6
ЗАДАНИЕ № 2.....	11
Алгоритм обучения линейного перцептрона .....	11
ЗАДАНИЕ № 3.....	19
Алгоритм обучения ядерного перцептрона.....	19
ЗАДАНИЕ № 4.....	27
Алгоритм обучения решающего дерева .....	27
Перечень использованных информационных ресурсов .....	34

## ОБЩИЕ СВЕДЕНИЯ И ВЫБОР ВАРИАНТА

Контрольная работа состоит из трех групп заданий: первая группа - написать ответ на теоретический вопрос, вторая группа – практические задания на освоение методов машинного обучения с учителем и третья группа – практические задания на освоение методов машинного обучения без учителя.

Варианты для задания первой группы приведены в таблице № 1.

### ЗАДАНИЯ ПЕРВОЙ ГРУППЫ

Таблица № 1. Распределение вопросов первого раздела контрольной работы по вариантам

ПРЕДПОСЛЕДНЯЯ ЦИФРА ЗАЧ. КН.	ПОСЛЕДНЯЯ ЦИФРА ЗАЧЁТНОЙ КНИЖКИ										
		0	1	2	3	4	5	6	7	8	9
	0	1	11	21	31	10	20	30	9	19	29
	1	2	12	22	1	11	21	31	10	20	30
	2	3	13	23	2	12	22	1	11	21	31
	3	4	14	24	3	13	23	2	12	22	1
	4	5	15	25	4	14	24	3	13	23	2
	5	6	16	26	5	15	25	4	14	24	3
	6	7	17	27	6	16	26	5	15	25	4
	7	8	18	28	7	17	27	6	16	26	5
	8	9	19	29	8	18	28	7	17	27	6
	9	10	20	30	9	19	29	8	18	28	7

Вопросы первого раздела контрольной работы:

- 1) Формальный алгоритм обучения перцептрона как линейного классификатора.
- 2) Нечистота разделения обучающих примеров и как она может быть вычислена.
- 3) Формальный алгоритм обучения дерева решений с алгоритмами подпрограмм.
- 4) Формальный алгоритм кластеризации К средних.
- 5) Формальный алгоритм обучения ядерного перцептрона.
- 6) Элементарное представление о решающем дереве и терминология деревьев.
- 7) Обобщенное описание алгоритма классификации по К ближайшим соседям.
- 8) Обобщенное расстояние Минковского, типовые нормы, свойства расстояния.

- 9) Одномерный метод наименьших квадратов и его интерпретация на многомерный случай (общее выражение без вывода). Недостатки такого подхода.
- 10) Обобщенное описание метода кластеризации K средних.
- 11) Признаки, классы и их кодирование в моделях решающих деревьев.
- 12) Ядерный трюк, что такое ядро и основных виды ядер, классификатор для ядерных методов.
- 13) Перцептрон в качестве линейного классификатора (общее описание) и формула коррекции вектора весов перцептрона.
- 14) Общие положения метода опорных векторов.
- 15) Общая методика обучения дерева решений.
- 16) Линейные модели для мультиклассовой классификации.
- 17) В чём суть метода опорных векторов с мягким зазором и его свойства.
- 18) Основные положения ядерных методов опорных векторов.
- 19) Определение системы поддержки принятия решений.
- 20) Общая структура СППР.
- 21) Классификация СППР по степени интеллектуальности и определение каждого вида СППР.
- 22) Количественная оценка правильности модели машинного обучения.
- 23) Общее и формальное определение машинного обучения.
- 24) Компоненты машинного обучения.
- 25) Классификация моделей машинного обучения и особенности типов машинного обучения.
- 26) Укрупнённая классификация методов машинного обучения.
- 27) Общее представление о линейных моделях классификации.
- 28) Преимущества и недостатки деревьев решений.
- 29) Общее представление о метрическом пространстве.
- 30) Понятие обобщающей способности, переобучения и недообучения модели.
- 31) Понятие центроиды и медоиды, как они рассчитываются.

Варианты для заданий второй и третьей группы выполняются в виде программ, реализующих алгоритмы машинного обучения, на языке программирования Python. Данные задания являются общими (т. е. одни и те же для всех студентов группы).

## ЗАДАНИЯ ВТОРОЙ И ТРЕТЬЕЙ ГРУППЫ

### ЗАДАНИЕ № 1.

#### Алгоритм кластеризация методом $K$ средних с применением евклидова расстояния

Цель — реализация алгоритма кластеризации данных методом  $K$  средних, а также исследование обобщающей способности и точности данного алгоритма машинного обучения на различных примерах данных.

#### Задание

Имеется алгоритм машинного обучения, позволяющий выполнять кластеризацию данных методом  $K$  средних. Данный алгоритм строится на основе метрических моделей машинного обучения. В листинге 1.1 приведен псевдокод алгоритма кластеризации данных методом  $K$  средних.

Листинг 1.1 Алгоритм кластеризации данных методом  $K$  средних

**Вход:** данные  $x \in D$ , число кластеров  $K$ .

**Выход:** центроиды кластеров  $\mu_1, \mu_2, \dots, \mu_K$  и кластеры  $D_1, D_2, \dots, D_K$ .

*/\*Случайная инициализация  $\mu_1, \mu_2, \dots, \mu_K$ \*/*

**for**  $j \leftarrow 1$  **to**  $K$  **do**

$\mu_j \leftarrow \text{random}()$ ;

**end**

*/\*пересчёт  $\mu_1, \mu_2, \dots, \mu_K$  и разделение  $x \in D$  на кластеры  $D_1, D_2, \dots, D_K$ \*/*

**while**  $\mu_1, \mu_2, \dots, \mu_K$  не перестанут изменяться **do**

**for**  $x$  **in**  $D$  **do**

$D_j \leftarrow \{x \in D \mid x \text{ отнесена к кластеру } j \Leftrightarrow \arg\min_j \text{Dis}_2(x, \mu_j)\};$

**end**

$\mu_j \leftarrow \frac{1}{|D_j|} \sum_{x \in D_j} x;$

**end**

**return**  $\mu_1, \mu_2, \dots, \mu_K, D_1, D_2, \dots, D_K;$

**(Основное задание: 65 баллов)**

1) Необходимо реализовать алгоритм кластеризации данных методом  $K$  средних на языке программирования **Python**, используя только типовые библиотеки: **numpy** и **matplotlib**.

2) Проверить работу машинного кластеризатора на файлах с данными (файлы с данными взять у преподавателя):

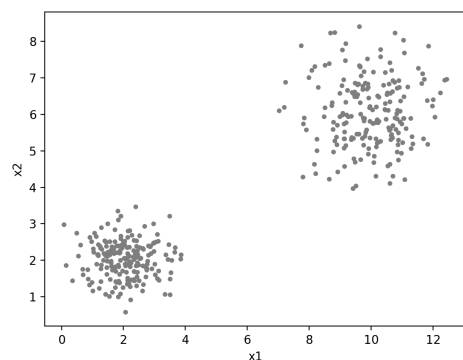
**lab\_metric\_data\_K\_2.npy** – два кластера  $K = 2$ ;

**lab\_metric\_data\_K\_3.npy** – три кластера  $K = 3$ , рекомендуется отлаживать алгоритм на данном наборе данных.

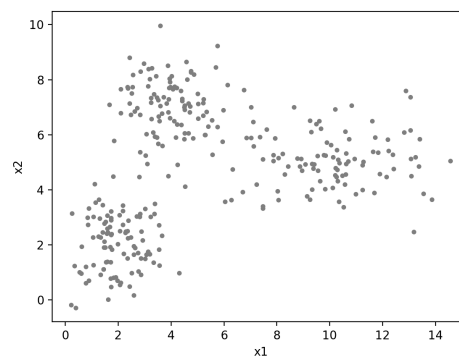
**lab\_metric\_data\_K\_5.npy** – пять кластеров  $K = 5$ .

Файлы **lab\_metric\_data\_K\_l.npy** ( $l = 2, 3, 5$ ) являются массивами библиотеки **numpy** для их считывания рекомендуется применять функцию **numpy.load()**.

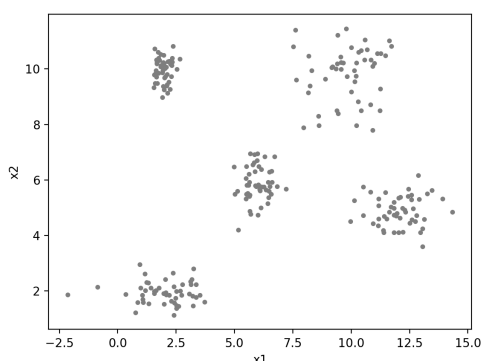
Считанные данные должны выглядеть как показано на рисунке 1.1.



(a)



(б)



(в)

Рис. 1.1 – Исходные данные:

(а) – `lab_metric_data_K_2.npy`; (б) – `lab_metric_data_K_3.npy`; (в) – `lab_metric_data_K_5.npy`

3) По каждому набору данных необходимо показать:

а) первоначальное расположение центройд кластеров  $\mu_1, \mu_2, \dots, \mu_K$ , которые заданы случайным образом. Пример показан на рисунке 1.2

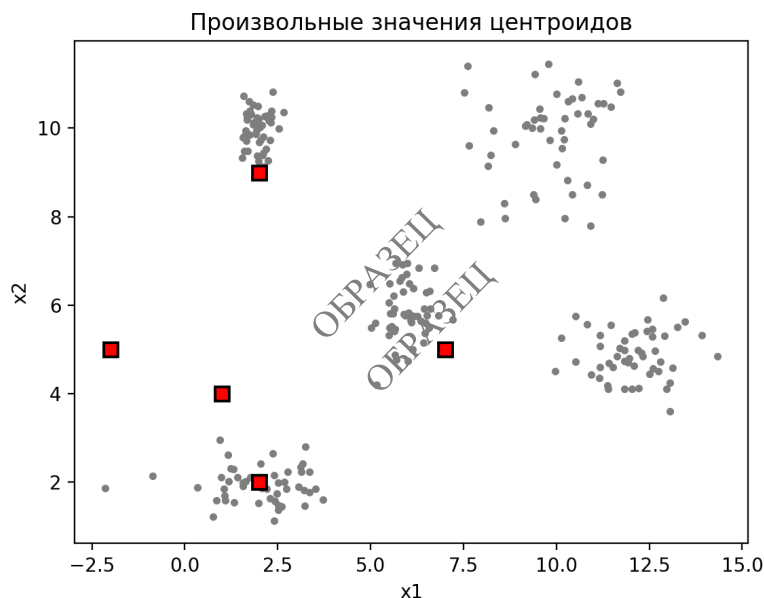


Рис. 1.2 – Первоначальное расположение центройд кластеров, заданное случайно

б) найденные алгоритмом расположения центройд кластеров  $\mu_1, \mu_2, \dots, \mu_K$ . Пример показан на рисунке 1.3

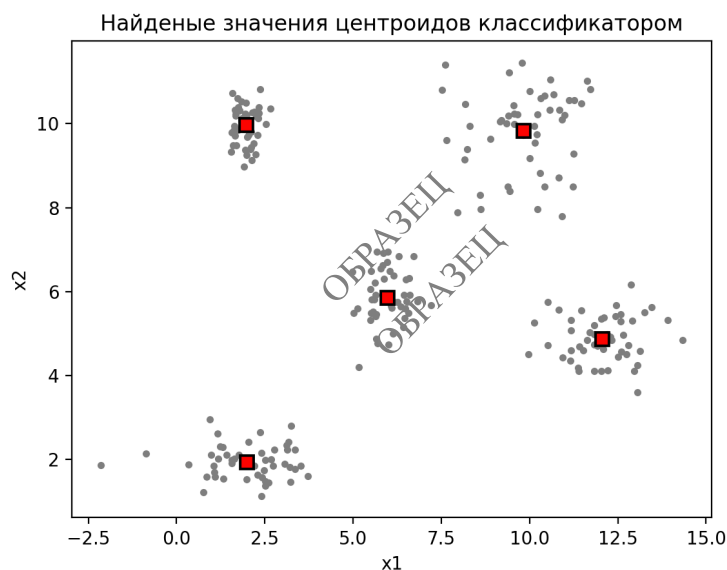


Рис. 1.3 – Найденные алгоритмом метода  $K$  средних расположение центройд кластеров



в) объекты, распределенные алгоритмом метода  $K$  средних по найденным кластерам  $D_1, D_2, \dots, D_K$ . Пример показан на рисунке 1.4

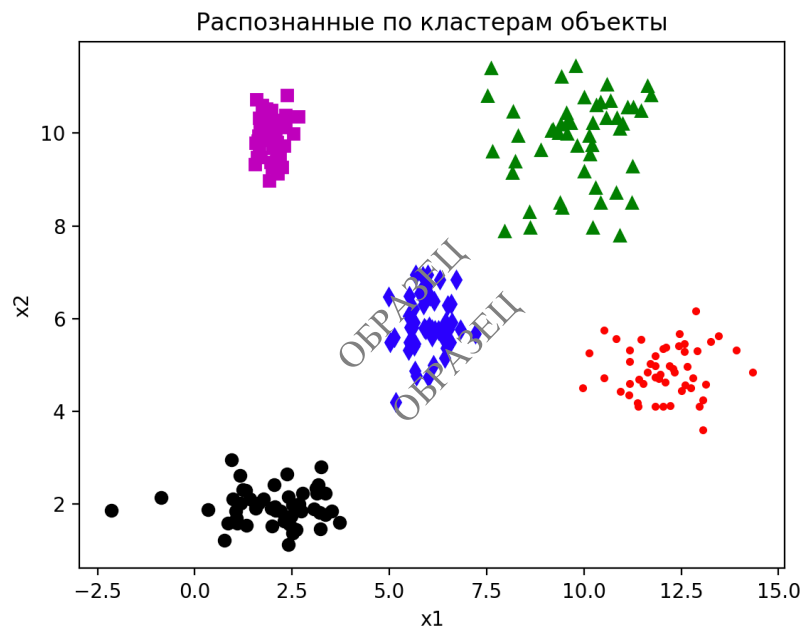


Рис. 1.4 – Объекты, распределенные алгоритмом машинного обучения по кластерам

**(Дополнительное задание № 01: +15 баллов => 65 + 15 = 80 баллов)**

Реализуйте код программы кластеризации таким образом, чтобы алгоритм метода  $K$  средних работал на любых двумерных данных с любым заданным количеством кластеров.

**(Дополнительное задание № 02: +20 баллов => 65 + 15 + 20 = 100 баллов)**

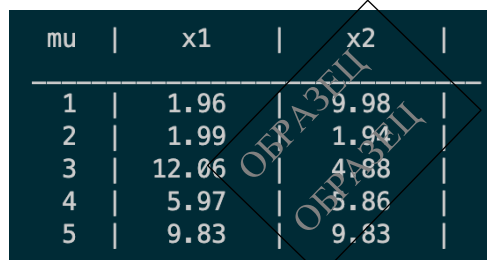
Реализуйте код программы кластеризации метода  $K$  средних в виде класса, конструктор которого должен принимать множество входных данных  $D$  и количество кластеров  $K$ , которые необходимо найти:

**kMeans = KMeans (D, K)**

имел метод **clusterisation()**, который бы возвращал координаты всех центроидов  $\mu_1, \mu_2, \dots, \mu_K$  и исходные данные разделенные по кластерам  $D_1, D_2, \dots, D_K$ .

Также необходимо перегрузить dunder-метод `__str__()` класса `KMeans`, которой бы возвращал в консоль найденные координаты центроидов (рис. 1.5)  $\mu_1, \mu_2, \dots, \mu_K$ :

```
print(kMeans)
```



mu	x1	x2
1	1.96	9.98
2	1.99	1.94
3	12.06	4.88
4	5.97	5.86
5	9.83	9.83

Рис. 1.5 – Таблица найденных координат центроидов  $\mu_1, \mu_2, \dots, \mu_K$

#### Отчет по заданию № 01:

- 1) листинг законченной программы для того уровня, на котором вы решили остановиться;
- 2) графики кластеризации данных для всех трех исходных данных (первоначальное расположение центроидов (см. рис. 1.2), найденное расположение центроидов (см. рис. 1.3), размеченные данные по кластерам (см. рис. 1.4)). Можно добавить графики с вариантами неправильной кластеризации, если такие будут;
- 3) для дополнительного задания № 02 необходимо добавить скриншот консоли с таблицей найденные координаты центроидов  $\mu_1, \mu_2, \dots, \mu_K$  (см. рис. 1.5).

## ЗАДАНИЕ № 2

### Алгоритм обучения линейного перцептрона

Цель — реализация и исследование работы алгоритма обучения перцептрона для линейной классификации объектов.

#### Задание

Имеется алгоритм машинного обучения, позволяющий выполнять линейную классификацию объектов. Данный алгоритм строится на основе обучения перцептрона (простейшей нейронной сети) выполнять линейное разделение двух классов объектов. Перцептрон перебирает обучающий набор данных (это метод машинного обучения с учителем), обновляя вектор весов всякий раз, как встречает неправильно классифицированный пример. В листинге 2.1 приведен псевдокод алгоритма обучения перцептрона как линейного классификатора.

#### (Основное задание: 65 баллов)

- 1) Необходимо реализовать алгоритм обучения перцептрона на языке программирования **Python**, используя только типовые библиотеки: **numpy** и **matplotlib**.
- 2) Для обучения машинного классификатора используются данные из файлов (файлы с данными взять у преподавателя): **lab\_perceptron\_data\_01.npy ... lab\_perceptron\_data\_05.npy**.

Замечания. Файлы **lab\_perceptron\_data\_01.npy**,  
**lab\_perceptron\_data\_02.npy**,  
**lab\_perceptron\_data\_05.npy**

содержат хорошо разделяемые классы данных, перцептрон можно будет обучить на одном и том же значении  $\eta$ .

## Листинг 2.1 Алгоритм обучения линейного перцептрона

**Вход:** помеченные входные данные  $x \in D$ ,  
скорость обучения  $\eta$ ,

величина смещения  $\vartheta$  <sup>\*)</sup>.

**Выход:** вектор весов  $w$  для классификатора  $\hat{y} = \text{sign}(w \cdot x)$ .

```
 $t \leftarrow 0;$  /*счетчик эпох*/
```

```
/*Инициализация вектора весов нулями, в общем можно инициализировать  
и другими значениями*/
```

```
 $w \leftarrow 0;$ 
```

```
/*флаг сходимости алгоритма и завершения главного цикла*/  
 $converged \leftarrow false;$ 
```

```
while  $converged = false$  do
```

```
/*если количество эпох больше какого-то числа, то алгоритм не  
сошёлся – решение не найдено*/
```

```
 $t \leftarrow t + 1;$ 
```

```
if  $t \geq 100$ 
```

```
then
```

```
     $error;$ 
```

```
end
```

```
/*коррекция вектора весов  $w$  перцептрона*/
```

```
 $converged \leftarrow true;$ 
```

```
for  $i \leftarrow 1$  in  $|D|$  do
```

```
    if  $y_i \cdot w \cdot x_i \leq 0$  /*т.е.  $\hat{y} \neq y^*$ */
```

```
    then
```

```
         $w \leftarrow w + \eta \cdot y_i \cdot x_i;$ 
```

```
         $\vartheta \leftarrow \vartheta + \eta \cdot y_i;$  *)
```

```
        /*меняли  $w$ , значит алгоритм еще не сошёлся*/
```

```
         $converged \leftarrow false;$ 
```

```
    end
```

```
end
```

```
end
```

```
return  $w, \vartheta$  *);
```

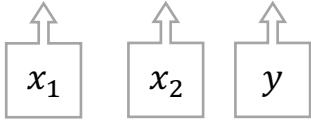
<sup>\*)</sup> Данные части алгоритма необходимы только для последующего графического приближенного отображения решающей границы, для визуального контроля классификации двумерных данных. Классический алгоритм обучения перцептрона используется только для отыскания вектора весов  $w$  классификатора.

Файл **lab\_perceptron\_data\_03.npy**, содержит плохо разделимые классы данных, поэтому, возможно, алгоритм не сойдётся.

Файл **lab\_perceptron\_data\_04.npy**, содержит близко расположенные друг к другу классы данных, поэтому скорее всего для их разделения потребуется отдельный подбор значения скорости обучения из диапазона  $0 < \eta \leq 1$  и для более правильного отображения решающей границы возможно придется подобрать величину смещения  $\vartheta$ .

**Формат данных.** Данные, содержащиеся в файлах **lab\_perceptron\_data\_XX.npy** представляют собой массив **numpy.array** массивов **numpy.array**:

```
array([
    array([1.23, 0.2, 1]),
    array([2.0, 1.5, 1]),
    .....
    array([0.8, 1.3, -1]),
    array([3.5, 2.3, -1])
])
```



$x_1, x_2$  – координаты точек по оси абсцисс и ординат соответственно, то есть значения двух признаков объекта. Так как рассматривается метод машинного обучения с учителем, а данные в файлах являются обучающими наборами, то каждый объект содержит метку класса  $y$ , которая принимает одно из двух значений  $y \in \{+1, -1\}$ . То есть объект относится к классу  $+1$  или классу  $-1$ .

Объекты класса  $+1$  можно показать на графике как “ $\circ$ ”, а объекты класса  $-1$  можно показать на графике как “ $\times$ ”, как показано на рисунке 2.1.

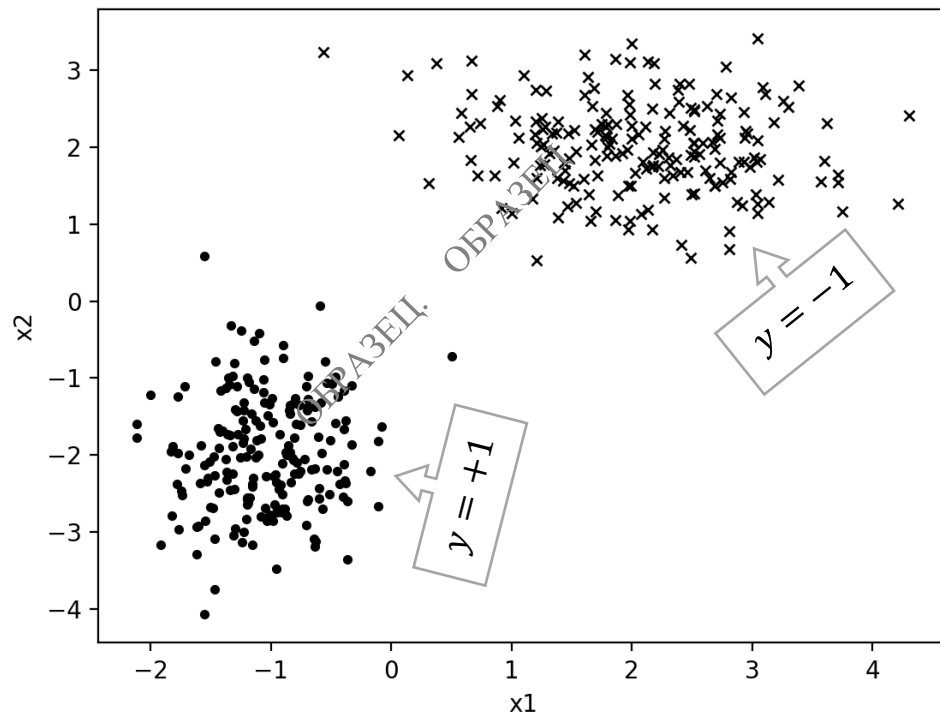


Рис. 2.1 – Пример считанных данных из файла **lab\_perceptron\_data\_01.npy**

3) Проверить работу классификатора, для этого необходимо:

а) на каждом графике, кроме того на котором алгоритм не сошёлся, показать графически решающую границу. Если алгоритм машинного обучения рассчитал вектор весов  $\mathbf{w} = \{\mathbf{w}_0 \ \mathbf{w}_1\}^T$  и скорректировал смещение  $\vartheta$ , то приблизительную решающую границу можно найти по выражению (2.1):

$$line(x_1) \approx \frac{-(w_0 \cdot x_1 + \vartheta)}{w_1} \quad (2.1)$$

На рисунке 2.2 показана приблизительная решающая граница для данных из файла **lab\_perceptron\_data\_01.npy**.

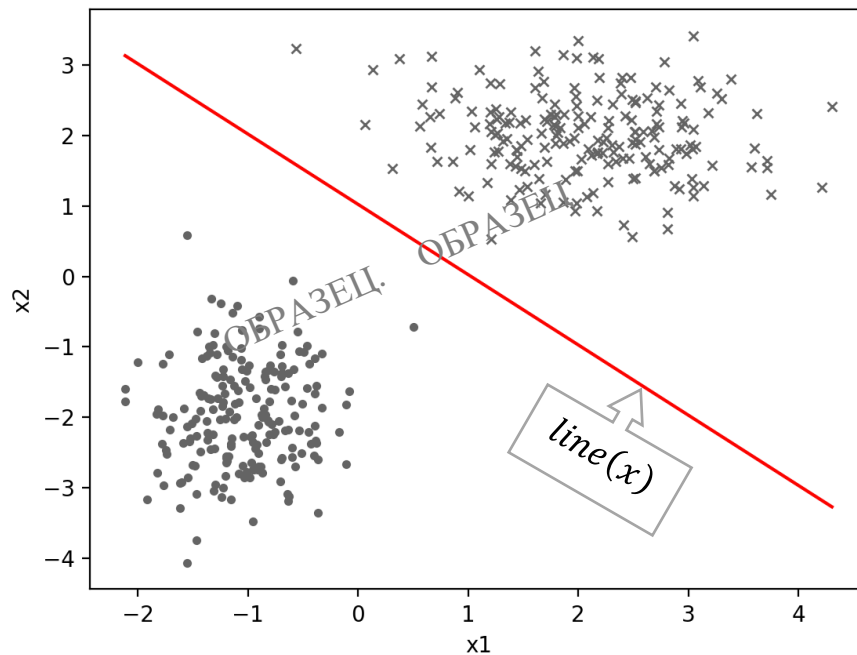


Рис. 2.2 – Приблизительная решающая граница для данных из файла **lab\_perceptron\_data\_01.npy**

б) задаться разными точками «сверху» и «снизу» приближенной решающей границы и по выражению линейного классификатора (2.2) найти класс, к которому относиться точка

$$\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x}) \quad (2.2)$$

где  $\mathbf{w} = \{\mathbf{w}_0 \quad \mathbf{w}_1\}^T$  – вектор весов, найденный перцептроном;

$\mathbf{x} = \{x_0 \quad x_1\}^T$  – координаты тестируемого объекта (точки).

На рисунке 2.3 показаны примеры тестирования. Для удобства классификатор возвращает значения классов не как  $y \in \{+1, -1\}$ , а как  $y \in \{°, \times\}$ .

На рисунке 2.3, а) показано совпадение найденного классификатором класса и графическим расположением объекта на плоскости. На рисунке 2.3, б) показана фиктивная ошибка классификатора, в действительности классификатор определил класс верно (объект ближе находится к классу « $\times$ »). Ошибка заключается в приближенном отображении решающей границы, которая сдвинута в сторону объектов класса « $\times$ ».

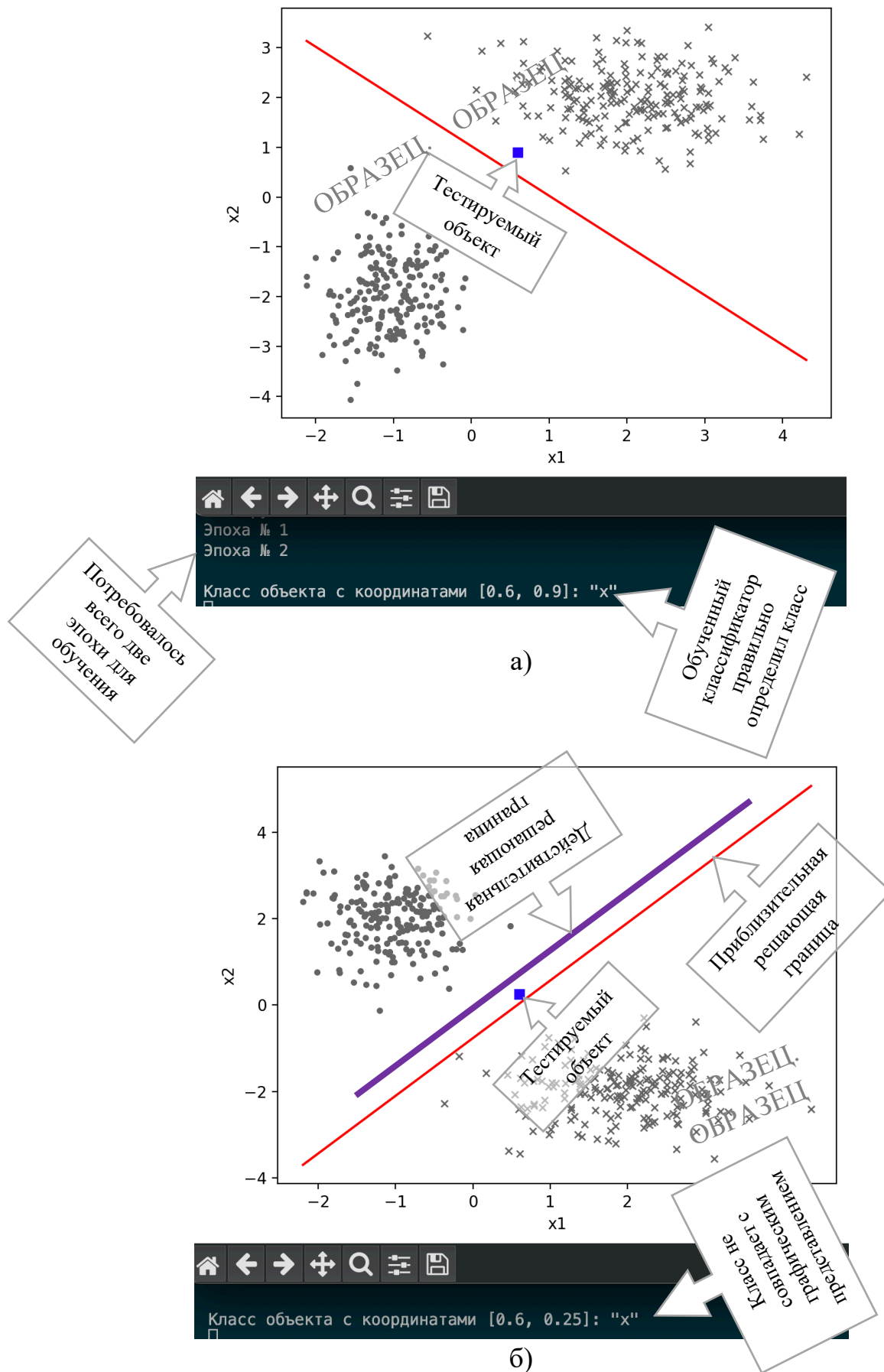


Рис. 2.3 – Примеры тестирования:  
 а) без ошибки классификации; б) с фиктивной ошибкой классификации



**(Дополнительное задание № 01: +15 баллов => 65 + 15 = 80 баллов)**

Реализуйте код программы линейного классификатора с обучением перцептрона, чтобы программа работала с любыми двумерных данных для двух классов (автоматически строила все необходимые графики, решающие границы, отмечала тестируемый объект на графике, относил тестируемый объект к соответствующему классу).

**(Дополнительное задание № 02: +20 баллов => 65 + 15 + 20 = 100 баллов)**

Реализуйте собственный класс обучения перцептрона, конструктор которого должен принимать массив обучающих данных  $D$ :

```
perceptron = Perceptron(D)
```

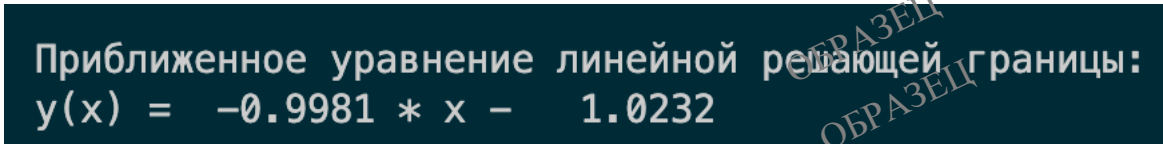
Класс должен реализовывать методы:

**training(eta, theta)** – для обучения перцептрона;

**classification(x)** – для выполнения классификации по найденному  $w$ .

Также необходимо перегрузить dunder-метод **\_\_str\_\_()** класса **Perceptron**, которой бы возвращал в консоль выражение приближенной решающей границы (рис. 2.4):

```
print(perceptron)
```



Приближенное уравнение линейной решающей границы:  
 $y(x) = -0.9981 * x - 1.0232$

Рис. 2.4 – Приближенное уравнение линейной решающей границы

## Отчет по заданию № 02:

- 1) листинг законченной программы для того уровня, на котором вы решили остановиться;
- 2) графики линейного разделения двух классов данных с приблизительными решающими границами, тестовыми объектами и результатами работы классификатора (см. рис. 2.3, пометки стрелками что есть что на графиках можно не делать);
- 3) для дополнительного задания № 02 необходимо добавить скриншот консоли с найденным выражением приблизительной решающей границы для пары вариантов (см. рис. 2.4).

## ЗАДАНИЕ № 3

### Алгоритм обучения ядерного перцептрона

Цель — реализация и исследование работы алгоритма обучения ядерного перцептрона с полиномиальным ядром.

#### Задание

Имеется алгоритм машинного обучения, позволяющий выполнять бинарную классификацию объектов с нелинейной решающей границей. Данный алгоритм строится на основе обучения ядерного перцептрона выполнять разделение двух классов объектов, имеющих нелинейную решающую границу. Перцептрон перебирает обучающий набор данных (это метод машинного обучения с учителем), обновляя вектор множителей Лагранжа. В листинге 3.1 приведен псевдокод алгоритма обучения ядерного перцептрона как бинарного классификатора.

#### (Основное задание: 65 баллов)

1) Необходимо реализовать алгоритм обучения ядерного перцептрона (см. листинг 3.1) на языке программирования **Python**, используя только типовые библиотеки: **numpy**, **matplotlib** и **csv** для считывания данных из файла в формате \*.csv.

2) Для обучения машинного классификатора используются данные из файла (файл с данными взять у преподавателя): **lab\_kernel\_perceptron\_data.csv**.

Объекты класса +1 можно показать на графике как “■”, а объекты класса -1 можно показать на графике как “○”, как показано на рисунке 3.1.

В качестве ядра перцептрона использовать полиномиальное ядро:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^p$$

### Листинг 3.1 Алгоритм обучения ядерного перцептрона

**Вход:** помеченные входные данные  $x \in D$ , ядерная функция  $\kappa(x_i, x_j)$ .  
**Выход:** множители  $\alpha_i$ , определяющих нелинейную решающую границу для классификатора  $\hat{y}(x_i) = \text{sign}(w(x_i))$ .

$\alpha_i \leftarrow 0$  для  $1 \leq i \leq |D|$ ; /\*инициализация вектора множителей нулями\*/  
 $\text{converged} \leftarrow \text{false}$ ; /\*флаг сходимости алгоритма и завершения цикла\*/

$t \leftarrow 0$ ; /\*счетчик эпох\*/

**while**  $\text{converged} = \text{false}$  **do**

/\*коррекция вектора множителей  $\alpha_i$ \*/

$\text{converged} \leftarrow \text{true}$ ;

/\*если количество эпох больше какого-то числа, то алгоритм не сошёлся – прервать цикл\*/

$t \leftarrow t + 1$ ;

**if**  $t \geq 30$

**then**

**break**;

**end**

**for**  $i \leftarrow 1$  **in**  $|D|$  **do**

**if**  $y_i \cdot \left( \sum_{j=1}^{|D|} \alpha_j y_j \cdot \kappa(x_i, x_j) \right) \leq 0$

**then**

$\alpha_i \leftarrow \alpha_i + 1$ ;

/\*меняли  $\alpha_i$ , значит алгоритм еще не сошёлся\*/

$\text{converged} \leftarrow \text{false}$ ;

**end**

**end**

**end**

**return**  $\alpha_i$ ;

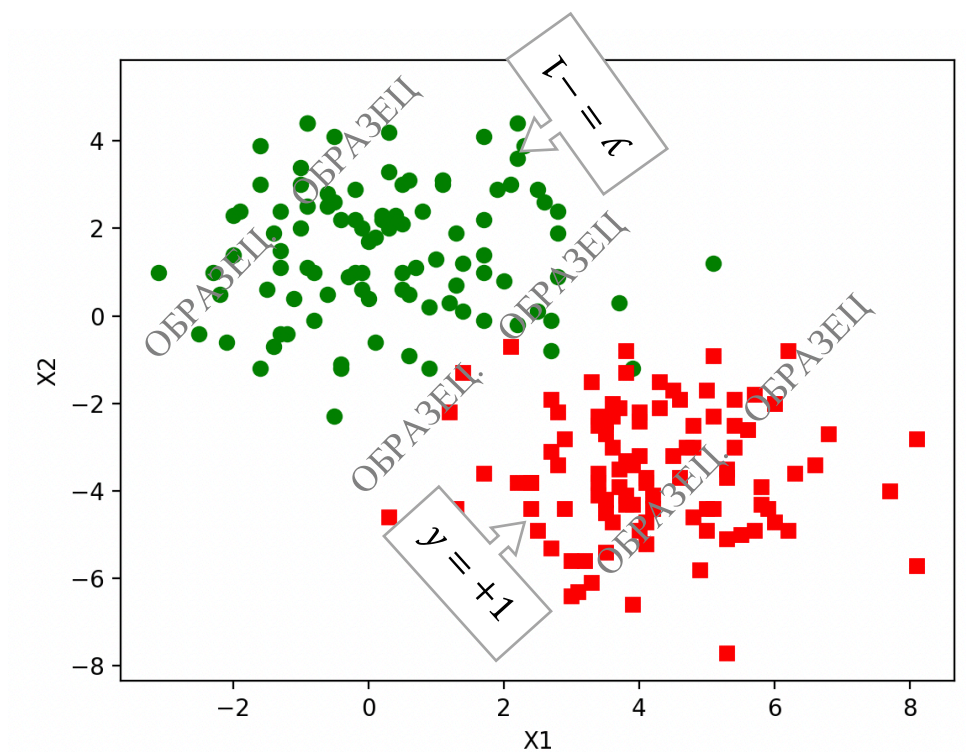


Рис. 3.1 – Отображение данных их файла **lab\_kernel\_perceptron\_data.csv**

**Формат данных.** Данные, содержащиеся в файле **lab\_kernel\_perceptron\_data.csv** представляют собой столбцы с признаками объекта  $\mathbf{X} = \{x_1 \ x_2\}$  и меткой класса  $y \in \{-1 \ +1\}$  (см. рис. 3.2) (разделитель данных – символ `'\t'`).

lab_kernel_perceptron_data.csv			
1	x1	x2	y
2	-----		
3	-0.5	-2.3	-1
4	-2.2	0.5	-1
5	8.1	-2.8	1
6	1.1	3.1	-1
7	-0.1	2.0	-1
8	3.5	-4.5	1

Рис. 3.2 – Формат файла данных

3) Отладку и проверку алгоритма обучения персептрона можно выполнять со степенью полиномиального ядра  $p = 2$ :

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^2$$

При этом, параметры классифицируемого объекта (координаты) можно взять в глубине обучающих классов:

$\mathbf{X} = \{0,95 \quad 0,80\}$  – класс «-1» (класс «кружки»)

$\mathbf{X} = \{4,30 \quad -2,64\}$  – класс «+1» (класс «квадратики»)

Добиваясь от персептрона правильной классификации тестируемого объекта (см. рисунок 3.3).

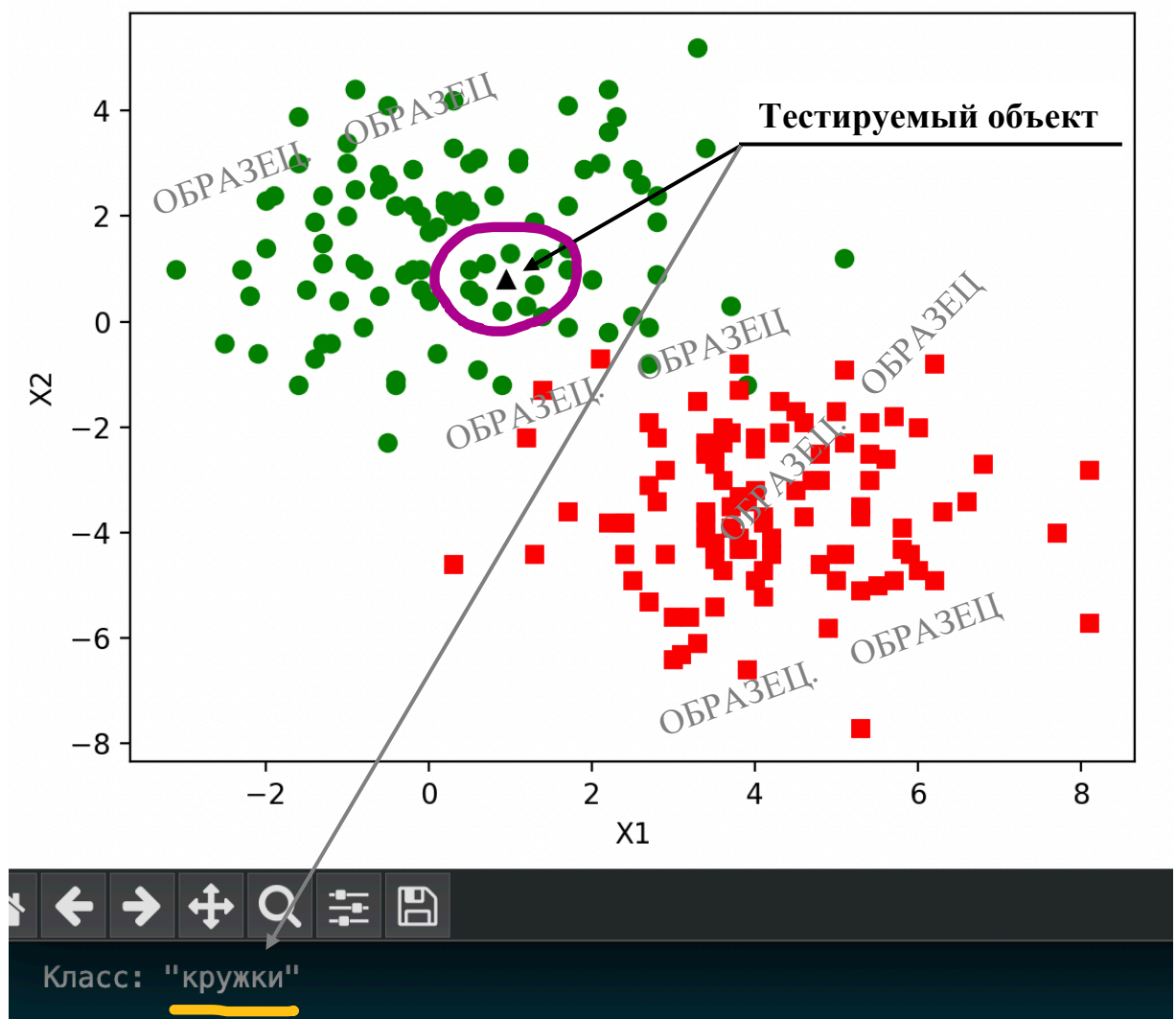


Рис. 3.3 – Пример классификации тестируемого объекта ядерным персептроном

**(Дополнительное задание № 01: +15 баллов => 65 + 15 = 80 баллов)**

Выполните исследование верности ядерного классификатора вблизи решающей границы при степенях  $p = 1$ ,  $p = 2$  и  $p = 3$  полиномиального ядра. Результаты исследования занесите в таблицу № 3.1.

- 1) Задайте степень полиномиального ядра  $p = 1$ .
- 2) Задайте признаки (координаты) для первого тестируемого объекта

$$\mathbf{X}_I = (2,36 \quad -0,16)$$

- 4) Выполните классификацию для оценки класса тестируемого объекта и выведите тестируемый объект на график (см. рисунок 3.4).

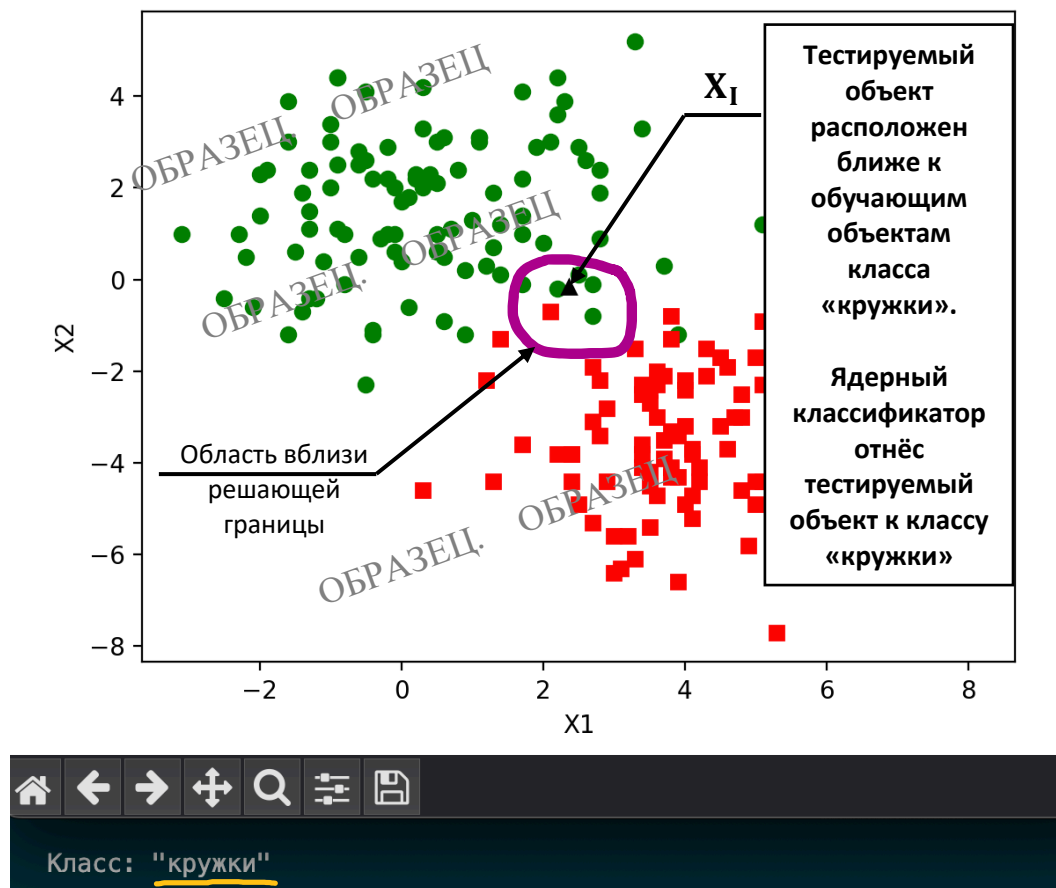


Рис. 3.4 – Исследование верности ядерного классификатора вблизи решающей границы

5) Результат, показанный на графике, соответствует реальному классу тестируемого объекта  $y(\mathbf{X}_I)$ , занесите его в соответствующую ячейку таблицы № 3.1. Результат, который выдал ядерный классификатор, соответствует оцененному классу  $\hat{y}(\mathbf{X}_I)$  тестируемого объекта, занесите его в соответствующую ячейку таблицы № 3.1.

6) Прodelайте тоже самое со вторым и третьим тестируемыми объектами

$$\mathbf{X}_{II} = (2,11 \quad -1,04)$$

$$\mathbf{X}_{III} = (2,57 \quad -0,74)$$

7) Когда заполните всю строчку таблицы № 3.1 для степени полиномиального ядра  $p = 1$ , поменяйте степень ядра на  $p = 2$  и выполните проверку ядерного классификатора снова для трёх тестируемых объектов  $\mathbf{X}_I$ ,  $\mathbf{X}_{II}$  и  $\mathbf{X}_{III}$ . Заполните строку  $p = 2$  таблицы № 3.1.

8) Задайте степень полиномиального ядра  $p = 3$  и выполните то же самое для трёх тестируемых объектов.

В некоторых случаях реальный класс тестируемого объекта и оцененный ядерным классификатором могут не совпадать.

Таблица № 3.1. Результаты исследования верности ядерного классификатора

Степень ядра	Класс для тестируемого объекта $\mathbf{X}_I$		Класс для тестируемого объекта $\mathbf{X}_{II}$		Класс для тестируемого объекта $\mathbf{X}_{III}$	
	реальный $y(\mathbf{X}_I)$	оцененный $\hat{y}(\mathbf{X}_I)$	реальный $y(\mathbf{X}_{II})$	оцененный $\hat{y}(\mathbf{X}_{II})$	реальный $y(\mathbf{X}_{III})$	оцененный $\hat{y}(\mathbf{X}_{III})$
$p = 1$	«кружки»	«кружки»				
$p = 2$						
$p = 3$						



**(Дополнительное задание № 02: +20 баллов => 65 + 15 + 20 = 100 баллов)**

Реализуйте собственный класс обучения ядерного перцептрона, конструктор которого должен принимать массив обучающих данных  $D$ , их метки и степень полиномиального ядра:

```
kernel_perceptron = KernelPerceptron(X, y,  
kernel_power)
```

Класс должен реализовывать методы и свойства:

**training()** – для обучения ядерного перцептрона;

**classification(x)** – для выполнения классификации тестируемого объекта;

**count\_of\_eras** – количество эр, за которое был обучен ядерный перцептрон.

Реализуйте вывод работы класса в консоль, в которой тип класса будет обрисовываться как показано на рисунке 3.5.



Рис. 3.5 – Пример вывода работы класса в консоль с отрисовкой класса

### Отчет по заданию № 03:

- 1) листинг законченной программы для того уровня, на котором вы решили остановиться;
- 2) графики классификации тестируемого объекта для каждого этапа выполнения лабораторной работы (см. рис. 3.3 и рис. 3.4, если делаете дополнительное задание № 01);
- 3) для дополнительного задания № 02 необходимо добавить скриншот графика и консоли демонстрирующих работу класса (см. рис. 3.5).

## ЗАДАНИЕ № 4

### Алгоритм обучения решающего дерева

Цель — реализация и исследование работы алгоритма обучения решающего дерева.

#### Задание

Требуется реализовать алгоритм машинного обучения решающего дерева, которое должно выполнять бинарную классификацию объектов. Данный алгоритм строится на основе последовательного разделения множества размеченных объектов на подмножества для формирования логических выражений отнесения тестового объекта к соответствующему классу по совокупности входных данных. Эти логические (конъюнктивные) выражения представлены в виде дерева – решающего дерева.

Выполнить машинное обучение решающего дерева означает получить древовидную структуру признаков и фактов из размеченного обучающего набора, который представлен в виде таблицы. Для машинного обучения решающего дерева требуется реализовать соответствующий алгоритм.

#### **(Основное задание: 65 баллов)**

1) Необходимо реализовать алгоритм обучения решающего дерева (см. листинг 4.1) на языке программирования **Python**. Из библиотек может потребоваться библиотека для визуализации полученного решающего дерева, например, **bigtree**, которая позволяет выводить древовидные структуры в консоль. Так же для считывания исходных данных из файла в формате \*.csv потребуются библиотека **csv**. считывания данных из файла в формате \*.csv.

## Листинг 4.1 Алгоритм обучения решающего дерева

**Сигнатура функции:** *GrowTree(D, F)*

**Вход:** помеченные входные данные  $D$ , множество признаков  $F$ .

**Выход:** дерево признаков (узлов)  $T$  с помеченными листьями  $L$ .

```

 $T \leftarrow \text{linkedlist}(\{\});$  /*связанный список структур*/
 $L \leftarrow \text{Label}(D);$  /*возвращаем класс объектов для множества*/
If  $L = \text{None}$  then /*класс для множества не определён*/
     $S \leftarrow \text{BestSplit}(D, F);$  /*возвращает наиболее чистый признак для разделения
множества  $D$ */
    If  $S = \text{None}$  then return; /*все признаки уже рассмотрены => ошибка и выход */

     $T \leftarrow \text{linkedlist}(S);$  /*добавили узел в дерево к родительскому узлу*/

    разделяем множество  $D$  на подмножества  $D_i$  в /*требуется реализовать*/
    соответствии с литералами в  $S$ ;

    for  $D_i$  in  $D$  do
         $\text{GrowTree}(D_i, F);$  /*рекурсивный вызов функции*/
    end
else
     $T \leftarrow \text{linkedlist}(L);$  /*добавили лист к родительскому узлу*/
end
return  $T$ ; /*дерево построено*/

```

**Сигнатура функции:** *BestSplit(D, F)*

**Вход:** помеченные входные данные  $D$ , множество признаков  $F$ .

**Выход:** признак  $f$ , по которому выполнять разделение.

```

 $I_{min} \leftarrow 1;$ 
 $f_{best} \leftarrow \text{None};$ 
for  $f$  in  $F$  do
    /*требуется реализовать*/
    разделить  $D$  на подмножества  $D_1, \dots, D_m$  согласно значениям  $v_j$  признака  $f$ ;
    /*т. е. необходимо для каждого значения признака найти разделение
 $f = [v_1, v_2, v_3] \Rightarrow [n_1+, n_1-] [n_2+, n_2-] [n_3+, n_3-]$ 
(Длина = {1, 2, 3} => [3+, 1-] [1+, 0] [1+, 4-])
*/
    If  $\text{Imp}(\{D_1, D_2, \dots, D_m\}) < I_{min}$  then
         $I_{min} \leftarrow \text{Imp}(\{D_1, D_2, \dots, D_m\});$ 
         $f_{best} \leftarrow f;$ 
    end
end
return  $f_{best};$ 

```

т.к. необходимо сохранять признаки по которым ранее было выполнено разделение, то могут быть значения типа  $n(+) = 0, n(-) = 0 \Rightarrow n(+) + n(-) = 0 \Rightarrow$  деление на нуль в таких случаях требуется пропуск итерации

Сигнатура функции: *Label(D)*

Вход: помеченные входные данные *D*.

Выход: возвращает один из классов или *None*.

for  $D_i$  in *D* do

/\*проверяем класс каждого  $D_i$  объекта с классов первого объекта  $D_1$ \*/

If  $\hat{c}(D_i) \neq \hat{c}(D_1)$  then

return *None*; /\*есть хотя бы одни  $D_i$  с классом отличным от других объектов из  $D^*$ \*/

end

end

return  $\hat{c}(D_1)$ ; /\*класс для объектов всего множества определён\*/

2) Для обучения решающего дерева используются данные из файла (файл с данными взять у преподавателя): **lab\_decision\_tree.csv**.

Легенда данных, следующая: у страхового агента, занимающегося страховкой транспортных средств есть некоторая статистическая информация, которую он накопил в результате своей деятельности. Эта информация представляет собой набор данных по страховке транспортных средств клиентов или отказах в страховке. В таблице 4.1 представлены эти статистические данные.

Таблица № 4.1. Статистические данные страхового агента

Возраст водителя, лет	Место эксплуатации ТС	Сведения о ДТП за последний год	Стаж вождения, лет	Тип авто	Решение о выдаче страховки
больше 40	город	есть	больше 10	обычное	страховать
больше 40	город	есть	меньше 10	обычное	отказать
больше 40	город	нет	меньше 10	обычное	страховать
меньше 40	город	нет	меньше 10	обычное	страховать
меньше 40	с/х местность	нет	меньше 10	обычное	отказать
меньше 40	город	нет	больше 10	спортивное	страховать
больше 40	с/х местность	есть	больше 10	обычное	страховать
меньше 40	с/х местность	есть	меньше 10	спортивное	отказать
больше 40	город	нет	больше 10	спортивное	страховать
меньше 40	с/х местность	есть	меньше 10	обычное	отказать

Для обучения решающего дерева, с последующей программной реализацией алгоритма машинного обучения, данные из таблицы № 4.1 были закодированы следующим образом:

Названия признаков:

Возраст водителя

⇒ age

Место эксплуатации ТС

⇒ location

Сведения о ДТП за последний год

⇒ accident

Стаж вождения

⇒ experience

Тип авто ⇒ typecar  
Решение о выдаче страховки ⇒ y

Коды значений признаков:

**Возраст водителя, лет**

больше 40 ⇒ 1  
меньше 40 ⇒ 0

**Место эксплуатации ТС**

город ⇒ 1  
с/х местность ⇒ 0

**Сведения о ДТП за последний год**

есть ⇒ 1  
нет ⇒ 0

**Стаж вождения, лет**

больше 10 ⇒ 1  
меньше 10 ⇒ 0

**Тип авто**

спортивное ⇒ 1  
обычное ⇒ 0

**Решение о выдаче страховки**

страховать ⇒ 1  
отказать ⇒ 0

После замены признаков и их значений кодами, таблица № 4.1 преобразуется к виду, показанному в таблице № 4.2.

Таблица № 4.2. Статистические данные страхового агента  
в закодированном виде

age	location	accident	experience	typecar	y
1	1	1	1	0	1
1	1	1	0	0	0
1	1	0	0	0	1
0	1	0	0	0	1
0	0	0	0	0	0
0	1	0	1	1	1
1	0	1	1	0	1
0	0	1	0	1	0
1	1	0	1	1	1
0	0	1	0	0	0

Данные из таблицы № 4.2 приведены в файле **lab\_decision\_tree.csv**.

В результате обучения должно быть построено решающее дерево на основе исходных данных. Один из примеров решающего дерева показан на рисунке 4.1. Для отображения структуры полученного решающего дерева использованы функции `dict_to_tree()` и `print_tree()` библиотеки `bigtree`.

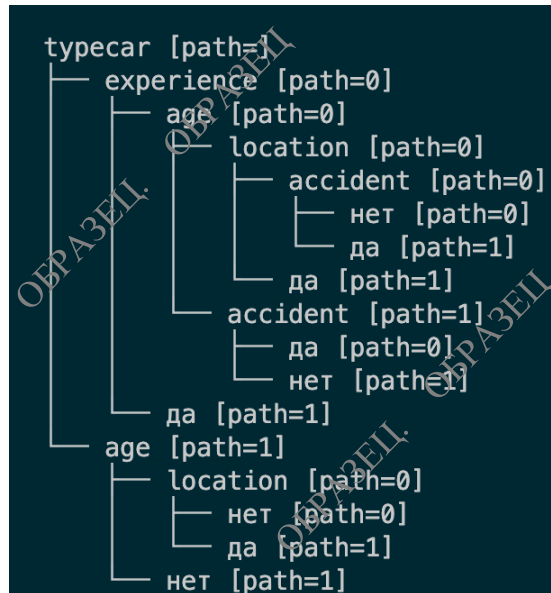


Рис. 4.1 – Пример построенного решающего дерева в консоли, которое получено в результате выполнения алгоритма машинного обучения

**(Дополнительное задание № 01: +15 баллов => 65 + 15 = 80 баллов)**

1) На основании полученного решающего дерева реализуйте программу, которая будет классифицировать входные данные. И выполните проверку на одной или двух записях из файла размеченных данных, как показано на рисунке 4.2.

age	location	accident	experience	typecar	y
меньше 40 лет	город	ДТП не было	меньше 10 лет	обычное	страховать
0	1	0	0	0	1

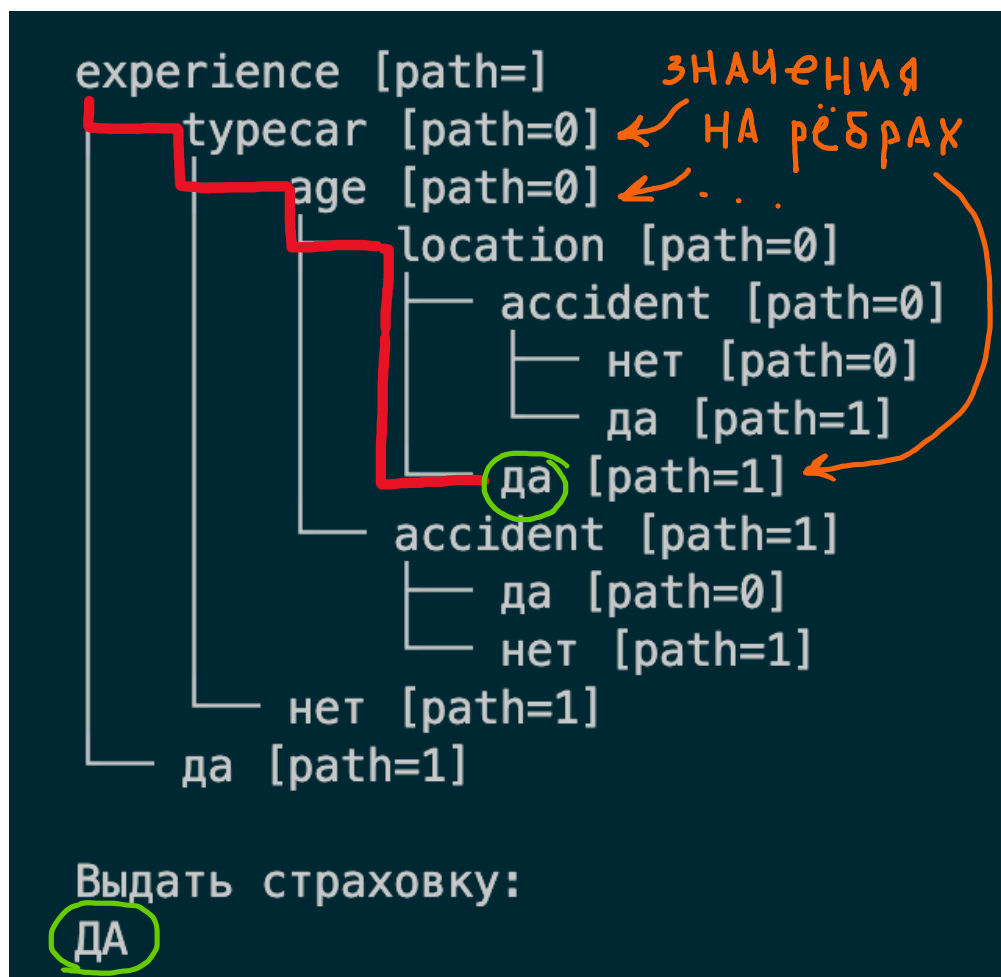


Рис. 4.2 – Проверка работы классификатора на обучающих данных

2) Измените только одно значение в исходном файле, то есть поменяйте **1** на **0** или **0** на **1** на пересечении любой строки и столбца, заново обучите решающее дерево и сравните его структуру с предыдущей. Ответьте на вопрос: «Изменилась ли структура дерева?». И если структура дерева изменилась, то как Вы думаете почему? Напишите это в отчёте лабораторной работы.



**(Дополнительное задание № 02: +20 баллов => 65 + 15 + 20 = 100 баллов)**

Реализуйте собственный класс обучения решающего дерева, конструктор которого должен принимать массив обучающих данных  $D$ , множество признаков  $F$ .

Класс должен реализовывать метод **training()** – для обучения решающего дерева, который будет возвращать структуру обученного дерева.

Отчет по заданию № 04:

- 1) листинг законченной программы для того уровня, на котором вы решили остановиться;
- 2) вывод структуры дерева в консоль или в виде графика, в зависимости от того, как вы решите визуализировать обученное решающее дерево (например, см. рис. 4.1);
- 3) для дополнительного задания № 01 необходимо добавить скриншот, показывающий работу классификатора и проверку его работы по решающему дереву (см. рис. 4.2). А также написать ответы на вопросы в данном задании.
- 4) для дополнительного задания № 02 необходимо реализовать алгоритм обучения решающего дерева в виде класса (в принципе это пункт перекрывается п. 1 требований к отчёту).

## Перечень использованных информационных ресурсов

1. Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2015. – 400 с.: ил.
2. Сузи, Р.А. Язык программирования Python: учеб. пособие, М.: Интернет-Ун-т Информ. Технологий: Бином. Лаборатория знаний, 2006.
3. Буйначев, С.К., Боклаг, Н.Ю. Основы программирования на языке Python: учебное пособие, Екатеринбург: Издательство Уральского университета, 2014.
4. Сузи, Р.А. Язык программирования Python: учебное пособие, М.: Интернет- Университет Информационных Технологий (ИНТУИТ), 2016
5. Уэс, Маккинли Python и анализ данных: практическое пособие, Саратов: Профобразование, 2017.
6. Балджы, А.С., Хрипунова, М.Б. Математика на Python: учебно-методическое пособие, М.: Прометей, 2018.
7. Гуриков С. Р. Основы алгоритмизации и программирования на Python: учебное пособие, М.: Издательство "ФОРУМ", 2017.
8. Жуков Р. А. Язык программирования Python: практикум: Учебное пособие, М.: ООО "Научно- издательский центр ИНФРА-М", 2020.