**Министерство науки и высшего образования РФ**Государственное образовательное учреждение высшего профессионального   
образования **«Тихоокеанский Государственный университет»**

Факультет компьютерных и фундаментальных наук

Кафедра ПОВТАС

**Лабораторная работа №4**

по дисциплине: «Методы машинного обучения»

на тему: «Реализация наивного байесовского классификатора»  
Вариант №4

Выполнил: студент группы ПИИ(м)-21

Забавин А.С.

Проверил: преподаватель кафедры ПОВТАС

Тормозов В.С.

# Постановка задачи

**Цель работы**: научиться строить наивный байесовский классификатор и с его помощью выполнять бинарную классификацию образов.

**Задания на лабораторную работу** (5 вариант)

1. Необходимо построить (реализовать на языке Python) наивный байесовский классификатор на основе, следующих данных обучающей выборки (для своего варианта):

[http://tk.ulstu.ru/files/iris data.py](http://tk.ulstu.ru/files/iris%20data.py)

Полагать, что признаки независимы и распределены по гауссовскому закону (нормальной плотности распределения вероятностей).

1. Для данной обучающей выборки подсчитать число и процент неверных классификаций.
2. Отобразить обучающую выборку в виде графика точек на плоскости (объекты разных классов должны быть иметь разные маркеры и цвет).

# Краткая теория

Плотность распределения случайной величины в соответствии с теоремой Байеса определяется выражением:

,

* – априорное распределение (доля соответствующего класса в обучающей выборке);
* – распределение образов для класса ;
* – распределение образов всех классов (без принадлежности к тому или иному классу);

Предположим, мы знаем величины и распределения Тогда, правило выбора класса (модель классификации) можно записать, следующим образом:

Главная проблема в реализации байесовского классификатора – это оценить условные плотности распределения вероятности . И, в общем виде, эта задача гораздо сложнее, чем построение классификатора с позиции выбора параметрической функции и последующего поиска вектора параметров :

При условии, что признаки независимы, функции правдоподобия классов представимы в виде:

Или, эквивалентный вывод часто делают по логарифму от произведения величин (уходят от произведений):

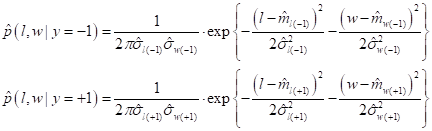
Часто в сторонних библиотеках алгоритм наивного байесовского классификатора реализуют с использованием гауссовских плотностей распределения вероятности. Конечно, это достаточно распространенный случай, но не всегда признаки подчиняются нормальному закону распределения. Это необходимо учитывать.

Если два зависимых признака и независимы и подчиняются нормальному распределению, справедливо:

* – математические ожидания и признаков;
* , – дисперсии и признаков.

Дисперсия определяется по формуле:

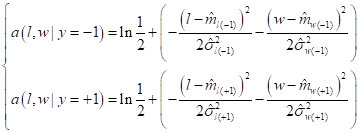
Перепишем, учитывая, что :



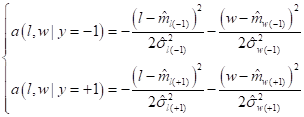
Априорные вероятности равны 0,5. Выполнив уход от произведений, построим эквивалентный классификатор по правилу:



Тогда получим:



Т.к. слагаемое одинаково у обоих классов, его можно не учитывать:



# Результаты работы

Работа была выполнена на языке программирования Python 3. Код программы представлена на Листинге 1.

import numpy as np

import matplotlib.pyplot as plt

import random

#===============================================================================

# Обучающая выборка

#===============================================================================

# вариант 5

data\_x = [(5.8, 1.2), (5.6, 1.5), (6.5, 1.5), (6.1, 1.3), (6.4, 1.3), (7.7, 2.0), (6.0, 1.8), (5.6, 1.3), (6.0, 1.6), (5.8, 1.9), (5.7, 2.0), (6.3, 1.5), (6.2, 1.8), (7.7, 2.3), (5.8, 1.2), (6.3, 1.8), (6.0, 1.0), (6.2, 1.3), (5.7, 1.3), (6.3, 1.9), (6.7, 2.5), (5.5, 1.2), (4.9, 1.0), (6.1, 1.4), (6.0, 1.6), (7.2, 2.5), (7.3, 1.8), (6.6, 1.4), (5.6, 2.0), (5.5, 1.0), (6.4, 2.2), (5.6, 1.3), (6.6, 1.3), (6.9, 2.1), (6.8, 2.1), (5.7, 1.3), (7.0, 1.4), (6.1, 1.4), (6.1, 1.8), (6.7, 1.7), (6.0, 1.5), (6.5, 1.8), (6.4, 1.5), (6.9, 1.5), (5.6, 1.3), (6.7, 1.4), (5.8, 1.9), (6.3, 1.3), (6.7, 2.1), (6.2, 2.3), (6.3, 2.4), (6.7, 1.8), (6.4, 2.3), (6.2, 1.5), (6.1, 1.4), (7.1, 2.1), (5.7, 1.0), (6.8, 1.4), (6.8, 2.3), (5.1, 1.1), (4.9, 1.7), (5.9, 1.8), (7.4, 1.9), (6.5, 2.0), (6.7, 1.5), (6.5, 2.0), (5.8, 1.0), (6.4, 2.1), (7.6, 2.1), (5.8, 2.4), (7.7, 2.2), (6.3, 1.5), (5.0, 1.0), (6.3, 1.6), (7.7, 2.3), (6.4, 1.9), (6.5, 2.2), (5.7, 1.2), (6.9, 2.3), (5.7, 1.3), (6.1, 1.2), (5.4, 1.5), (5.2, 1.4), (6.7, 2.3), (7.9, 2.0), (5.6, 1.1), (7.2, 1.8), (5.5, 1.3), (7.2, 1.6), (6.3, 2.5), (6.3, 1.8), (6.7, 2.4), (5.0, 1.0), (6.4, 1.8), (6.9, 2.3), (5.5, 1.3), (5.5, 1.1), (5.9, 1.5), (6.0, 1.5), (5.9, 1.8)]

data\_y = [-1, -1, -1, -1, -1, 1, 1, -1, -1, 1, 1, -1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, 1, 1, -1, 1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, -1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1]

#===============================================================================

# Очистка данных ===============

# Удалим дубликаты

data\_x\_plus\_y = zip(data\_x, data\_y)

data\_x\_plus\_y = np.array([[\*xy[0], xy[1]] for xy in data\_x\_plus\_y])

data\_x\_plus\_y = np.unique(data\_x\_plus\_y, axis=0)

data\_x = [list(xy)[:-1] for xy in data\_x\_plus\_y]

data\_y = [list(xy)[-1] for xy in data\_x\_plus\_y]

# ==============================

# Синоним: -1 = c1, +1 = c2

x\_train = np.array(data\_x)

y\_train = np.array(data\_y)

# Матожидание

mx1\_c1, mx2\_c1 = np.mean(x\_train[y\_train == -1], axis=0) # питон сахарок, предикат (y\_train == 1) возвращает selector (array\_like of bool) который может использоваться в \_\_getitem\_\_(). Это все считает матожидание от признаков определенного класса

mx1\_c2, mx2\_c2 = np.mean(x\_train[y\_train == 1], axis=0)

# Дисперсия

sx1\_c1, sx2\_c1 = np.var(x\_train[y\_train == -1], axis=0, ddof=1) # формула для вычисления дисперсии здесь немного другая (1/N)\*sum(...),

sx1\_c2, sx2\_c2 = np.var(x\_train[y\_train == 1], axis=0, ddof=1) # Правильная формула дисперсии (1/(N-1))\*sum((xi - mx)^2), Ddof нас спасет!

print(*f'Класс -1, Матожидание признака mx1={mx1\_c1}, mx2={mx2\_c1}, '*)

print(*f'Класс +1, Матожидание признака mx1={mx1\_c2}, mx2={mx2\_c2}, '*)

print(*f'Класс -1, Дисперсия признака sx1={sx1\_c1}, sx2={sx2\_c1}, '*)

print(*f'Класс +1, Дисперсия признака sx1={sx1\_c2}, sx2={sx2\_c2}, '*)

# модель на Плотность вероятности класса -1

def **a\_с1**(x): return -(x[0] - mx1\_c1) \*\* 2 / (2 \* sx1\_c1) - (x[1] - mx2\_c1) \*\* 2 / (2 \* sx2\_c1)

# модель на Плотность вероятности класса +1

def **a\_с2**(x): return -(x[0] - mx1\_c2) \*\* 2 / (2 \* sx1\_c2) - (x[1] - mx2\_c2) \*\* 2 / (2 \* sx2\_c2)

def **classify**(x):

*"""*

*Классификатор, -1, +1*

*"""*

return (-1 if int(np.argmax([a\_с1(x), a\_с2(x)])) == 0 else 1)

# Класифицированные образы

y\_classified = [(i, a\_с1(x), a\_с2(x), classify(x), ) for i, x in enumerate(x\_train)]

def **\_color\_by\_class**(class\_):

if class\_ == -1:

return *'green'*

return *'blue'*

# Построение графиков ----------------------------------------------------------

for class\_ in (-1, 1):

initial\_class\_points = x\_train[y\_train == class\_]

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))

fig.suptitle(*f'Наивный байесовский классификатор\n(вариант №5, размер выборки: {len(data\_x)})'*, fontsize=16)

ax[0].set\_title(*'Первоначальное распределение'*, color=*'black'*)

ax[0].scatter(

initial\_class\_points.transpose()[0],

initial\_class\_points.transpose()[1],

color=\_color\_by\_class(class\_), label=*f'Класс {class\_}'*,

alpha=1.0,

)

ax[0].tick\_params(labelcolor=*'indigo'*)

ax[0].legend()

y\_classified\_predicate = [(True if (y\_train[x[0]] == class\_ and x[3] == class\_) else False) for x in y\_classified]

classified\_class\_points = x\_train[y\_classified\_predicate]

y\_not\_classified\_predicate = [(True if (y\_train[x[0]] == class\_ and x[3] != class\_) else False) for x in y\_classified]

not\_classified\_class\_points = x\_train[y\_not\_classified\_predicate]

count\_of\_good\_defined = len(classified\_class\_points)

count\_of\_bad\_defined = len(not\_classified\_class\_points)

ax[1].set\_title(*"Распределение классификатора\n"* +

*f"{count\_of\_bad\_defined} ошибок = "* +

*f"{100 \* count\_of\_bad\_defined / len(data\_x)} %"*,

color=*'black'*

)

if count\_of\_good\_defined:

ax[1].scatter(

classified\_class\_points.transpose()[0],

classified\_class\_points.transpose()[1],

color=\_color\_by\_class(class\_), label=*f'Класс {class\_}'*,

alpha=1.0

)

if count\_of\_bad\_defined:

ax[1].scatter(

not\_classified\_class\_points.transpose()[0],

not\_classified\_class\_points.transpose()[1],

color=\_color\_by\_class(class\_),

label=*f"Ошибки"*,

alpha=0.2

)

for i, x in enumerate(not\_classified\_class\_points):

yc = y\_classified[i]

x\_offset = round(random.uniform(-40.0, 40.0), 3)

y\_offset = round(random.uniform(-50.0, 60.0), 3)

rand\_rad = round(random.uniform(-1.0, 1.0), 1)

ax[1].annotate(*f'i={yc[0]}\nс1,c2={yc[1]:.3f},{yc[2]:.3f}'*,

(x[0], x[1]),

textcoords=*"offset points"*,

xytext=(0.0 + x\_offset, 5.0 + y\_offset),

ha=*'center'*,

color=*'#4e084a'*, backgroundcolor=*"#cea7a72b"*,

arrowprops=dict(arrowstyle=*"->"*, connectionstyle=*f"arc3,rad={rand\_rad}"*)

)

ax[1].tick\_params(labelcolor=*'indigo'*)

ax[1].legend()

plt.show()

# ------------------------------------------------------------------------------

Листинг 1. Код программы main.py

Результат работы программы приведен на Рисунке 1. и Рисунке 2

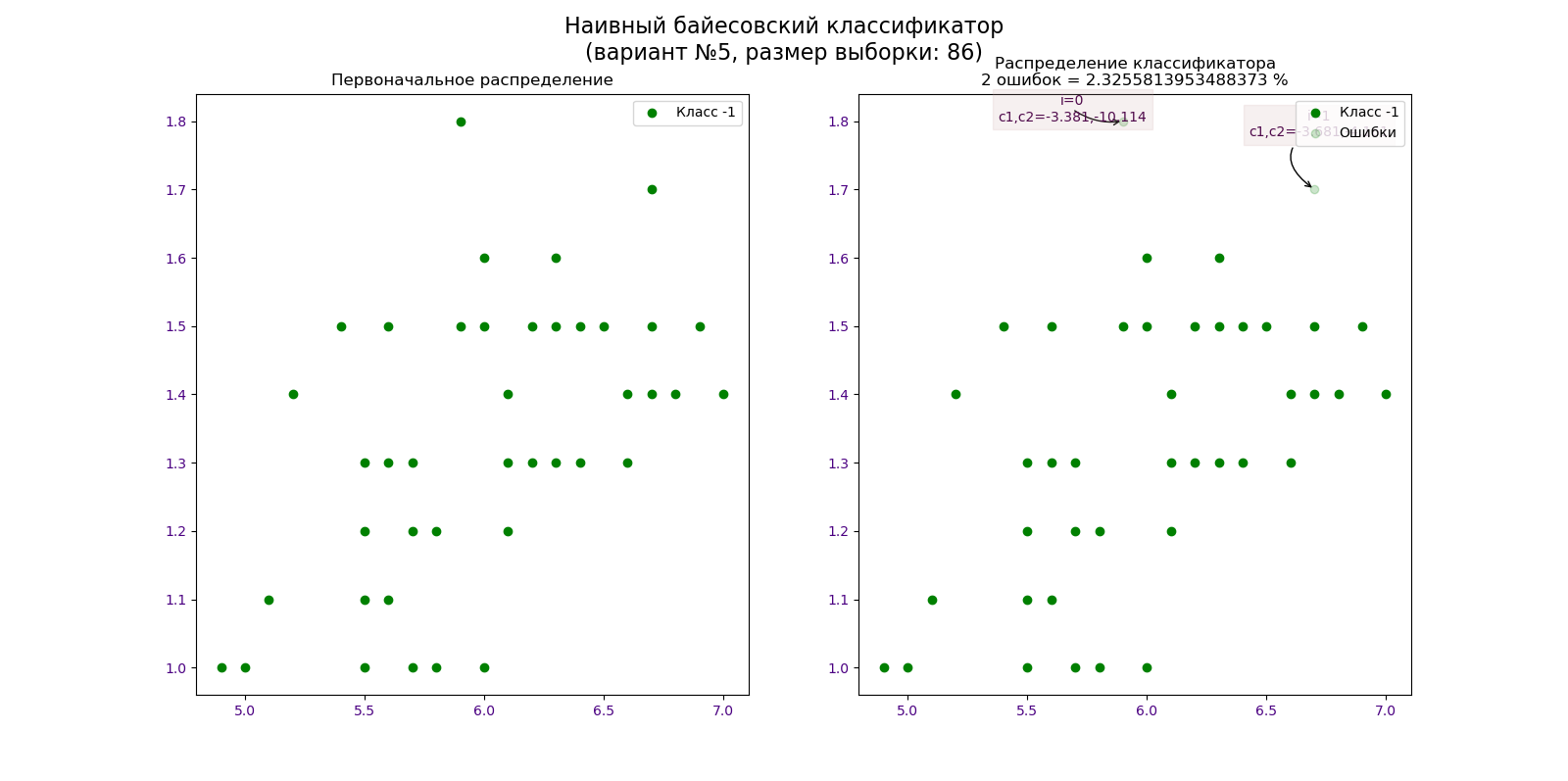


Рисунок 1. Результаты работы программы (Класс -1)

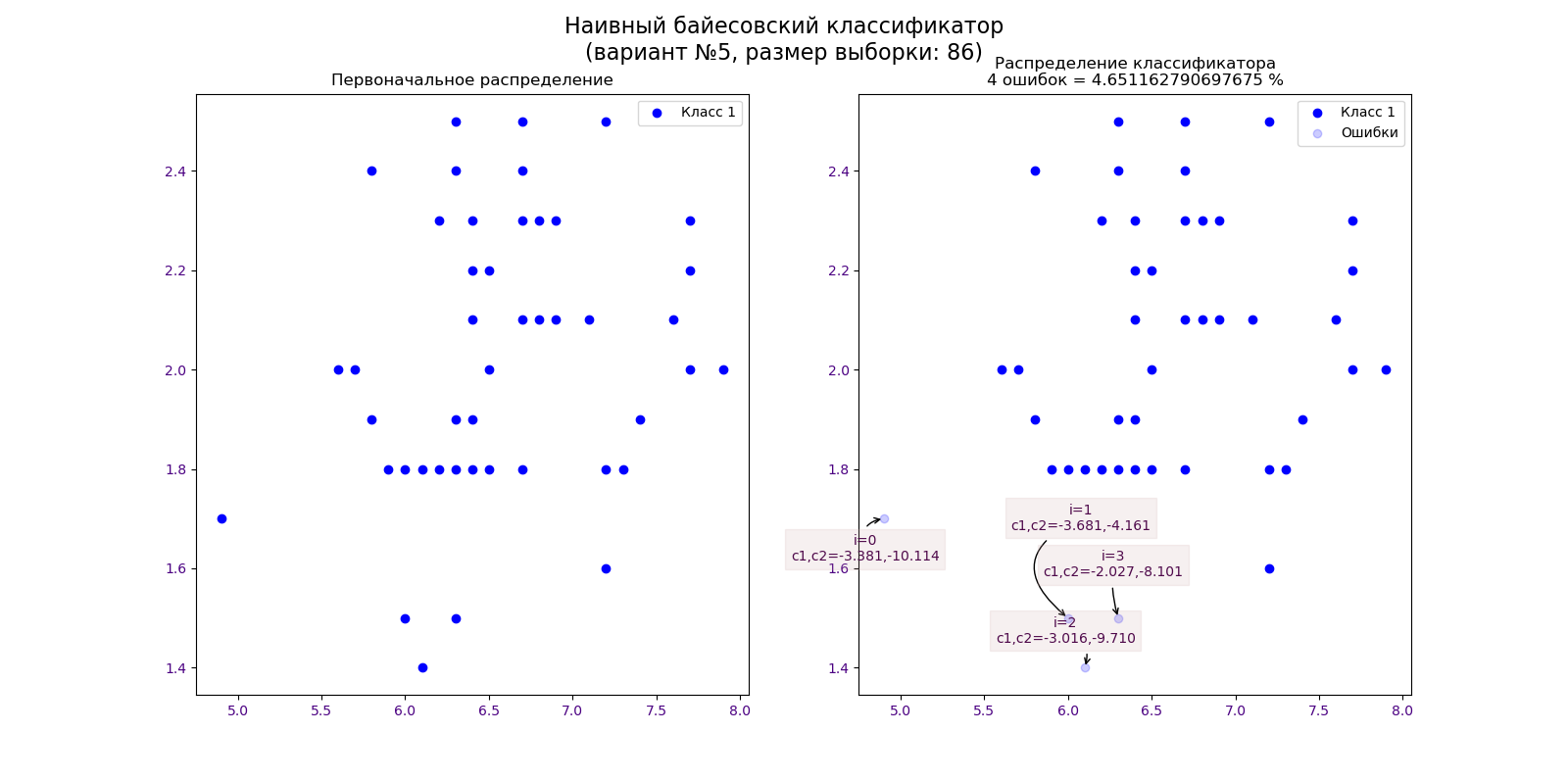


Рисунок 2. Результаты работы программы (Класс +1)

Класс -1, Матожидание признака mx1=5.995121951219512, mx2=1.3317073170731704,

Класс -1, Дисперсия признака sx1=0.28647560975609754, sx2=0.04271951219512193,

Класс +1, Матожидание признака mx1=6.582222222222221, mx2=2.0222222222222217,

Класс +1, Дисперсия признака sx1=0.4037676767676768, sx2=0.07904040404040404,

# Вывод

В ходе лабораторной работы был реализован наивный байесовский классификатор, определено число и процент ошибок, построены соответствующие графики.