**Федеральное агентство по образованию**Государственное образовательное учреждение высшего профессионального   
образования **«Тихоокеанский Государственный университет»**

Факультет компьютерных и фундаментальных наук

Кафедра ПОВТАС

**Лабораторная работа №4**

по дисциплине: «Архитектура систем ИИ»

на тему: «Точки (Р-модель распознавания). Используя ИНС»  
Вариант №4

Выполнил: студент группы ПИИ(м)-21

Забавин А.С.

Проверил: ст. преп. кафедры ПОВТАС

Тормозов В.С.

# Постановка задачи

Пусть образы объектов описываются группами из двух целочисленных параметров (x,y). Имеется два непересекающихся класса объектов. Требуется провести границу между классами. Способ построения разграничивающей прямой предлагается разработать самостоятельно.

**Исходные данные**

Два натуральных числа N1 – количество образцов из первого класса и N2 – количество образцов из второго класса. N1+N2 пар чисел (xk,yk) для образцов из первого и второго классов.

Требуется выполнить графическую иллюстрацию Р-модели.

**Замечание**

Точки разных классов могут задаваться пользователем произвольно или генерироваться автоматически. Для автоматического формирования наборов точек (xk,yk) каждого класса следует воспользоваться следующей информацией. Пусть в пространстве признаков R2 заданы два нормальных распределения с математическими ожиданиями (Mx1,My1) и (Mx2,My2) и дисперсиями σ1 и σ2.

Каждое из распределений задает один из классов объектов. Производится случайный выбор точек (объектов) и разыгрывается по заданным законам класс, в который они зачисляются. После того, как определены N1+N2 объектов, считаем, что исходная информация задана.

Таким образом, при разработке программы следует предусмотреть ввод пользователем величин N1, N2, Mx1, My1, Mx2, My2, σ1 и σ2.

# Краткая теория

Р-модель (модель разделения) характеризуется тем, что проводиться граница между классами в пространстве размерности . При построении информационного вектора исследуется положение объекта относительно данной границы. Сами объекты в этом случае рассматриваются как точки n-мерного пространства.

На Рис. 1а изображены объекты трех различных классов, между которыми проведены границы – прямые.

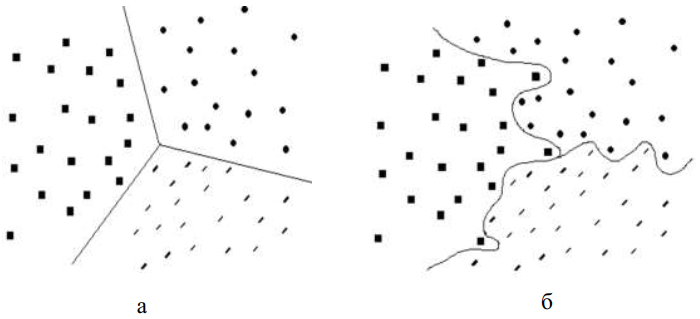


Рис. Пример разделения объектов 3-х классов

Не всегда взаимное расположение таково, что удается разделить классы прямыми линиями. В этом случае можно либо согласиться с возникающей погрешностью распознавания, либо проводить границы кривыми более высокого порядка (Рис. 1б).

При реализации Р-модели цель состоит в построении поверхностей, которые разделяли бы не только имеющиеся образцы, но и все остальные точки, принадлежащие классам. Иначе говоря, необходимо построить таких функции от векторов-образов объектов, которые принимали бы одинаковые значения для всех объектов одного класса и отличались от значений для объектов других классов. В связи с тем, что области не имеют общих точек, всегда существует целое множество таких разделяющих функций.

# Результаты работы

Работа была выполнена с помощью языка программирования Python версии 3 в составе дистрибутива ANACONDA (Miniconda3-py37\_4.12.0-Windows-x86\_64), для визуализации точек пакет matplotlib. Для построения нейронных сетей использовался пакет sklearn.neural\_network и tensorflow.keras .

В программе представлены три класса объектов:

**Image** – образ с произвольным набором признаков;

**ClassNormalCloud** - класс «облака» т.е. множества образов в пространстве размерности сгенерированных по закону нормального распределения (Гаусса), пользователь задает параметры множества при инициализации класса в подобном виде N=…, x={*'M'*: …, *'D'*: …}, y={*'M'*: …, *'D'*: …} , где N – число образов или величина множества, M – математическое ожидание по признаку, D – дисперсия по признаку. Исходя из аргументов при инициализации множество заполняется образами.

**NNCloudComparator** - «сравнитель» множеств, наследуемый от **CloudComparator** в ЛР№2 служебный класс имплементирующий различные операции между двумя множествами. В классе в дополнение к предыдущим методам переопределен конструктор и добавлены методы **nn\_fit()** - обучить ИНС-классификатор (MLPClassifier) предоставляемый sklearn по образцам из «обучающих множеств», и **nn\_get\_recognite\_images()**.

Классу классификатора заданы параметры наиболее подходящие для бинарной классификации точек в данной ЛР: функция активации - *relu*; количество скрытых слоев - 2 слоя по 2 нейрона, большие значения приводят к излишнему переобучению сети; оптимизатор весов *lbfgs* и альфа = 1e-5 .

Классификатор для обучения принимает на вход пару – набор признаков и соответственный набор значений принадлежности к классу (0/1).

Код представлен в листинге 1 (некоторые части сокращены «. . . . . . .», полный код программы в приложенном файле main.py)

class **NNCloudComparator**(CloudComparator):

*"""*

*Сравнитель облак образов (ИНС)*

*"""*

cloud1 = None

cloud2 = None

def **\_\_init\_\_**(*self*, cloud1: ClassNormalCloud, cloud2: ClassNormalCloud):

super().\_\_init\_\_(cloud1=cloud1, cloud2=cloud2)

*self*.nn\_classifier = MLPClassifier(solver=*'lbfgs'*, alpha=1e-5, hidden\_layer\_sizes=(2, 2), random\_state=1)

def **nn\_fit**(*self*):

cloud1klass = *self*.cloud1.klass

cloud2klass = *self*.cloud2.klass

def **ret\_features\_array**(im):

return np.array([getattr(im, x) for x in *self*.features\_names])

def **ret\_get\_01\_klass**(im):

klass = getattr(im, *'klass'*)

if klass == cloud1klass:

return 0

elif klass == cloud2klass:

return 1

X = list(map(ret\_features\_array, *self*.cloud1.\_images + *self*.cloud2.\_images))

y = list(map(ret\_get\_01\_klass, (*self*.cloud1.\_images + *self*.cloud2.\_images)))

*self*.nn\_classifier.fit(X, y)

def **nn\_get\_recognite\_images**(*self*, cloud3, copy\_image=False):

cloud1klass = *self*.cloud1.klass

cloud2klass = *self*.cloud2.klass

def **ret\_features\_array**(im):

return np.array([getattr(im, x) for x in *self*.features\_names])

def **actual\_class\_fro\_image**(klass01):

if klass01 == 0:

return cloud1klass

elif klass01 == 1:

return cloud2klass

imfeat = np.array(list(map(ret\_features\_array, cloud3.\_images)))

recognition\_result = *self*.nn\_classifier.predict(imfeat)

recognition\_result = list(map(actual\_class\_fro\_image, recognition\_result))

images = []

for image, newklass in zip(cloud3.\_images, recognition\_result):

iminst = Image(\*\*{f: getattr(f, image) for f in *self*.features\_names}) if copy\_image else image

setattr(iminst, *'klass'*, newklass)

images.append(iminst)

return images

if \_\_name\_\_ == *"\_\_main\_\_"*:

print(*'Введите N1: '*)

N1 = input()

N1 = int(N1) if N1 else 10000

print(*'Введите N2: '*)

N2 = input()

N2 = int(N2) if N2 else 10000

print(*'Введите Mx1: '*)

Mx1 = input()

Mx1 = int(Mx1) if Mx1 else 100

print(*'Введите My1: '*)

My1 = input()

My1 = int(My1) if My1 else 100

print(*'Введите Mx2: '*)

Mx2 = input()

Mx2 = int(Mx2) if Mx2 else 2500

print(*'Введите My2: '*)

My2 = input()

My2 = int(My2) if My2 else 110

print(*'Введите Dx1: '*)

Dx1 = input()

Dx1 = int(Dx1) if Dx1 else 10000

print(*'Введите Dy1: '*)

Dy1 = input()

Dy1 = int(Dy1) if Dy1 else 100000

print(*'Введите Dx2: '*)

Dx2 = input()

Dx2 = int(Dx2) if Dx2 else 10000

print(*'Введите Dy2: '*)

Dy2 = input()

Dy2 = int(Dy2) if Dy2 else 10000

print(*'Значения введены, программа расчитывает оптимальную линию...'*)

cloud1 = ClassNormalCloud(N1, x={*'M'*: Mx1, *'D'*: Dx1}, y={*'M'*: My1, *'D'*: Dy1}, klass=0)

cloud1.fill\_cloud\_Rn\_dimension()

cloud2 = ClassNormalCloud(N2, x={*'M'*: Mx2, *'D'*: Dx2}, y={*'M'*: My2, *'D'*: Dy2}, klass=1)

cloud2.fill\_cloud\_Rn\_dimension()

features\_x1 = list(itertools.chain(cloud1.get\_feature\_iterator(*'x'*)))

features\_y1 = list(itertools.chain(cloud1.get\_feature\_iterator(*'y'*)))

features\_x2 = list(itertools.chain(cloud2.get\_feature\_iterator(*'x'*)))

features\_y2 = list(itertools.chain(cloud2.get\_feature\_iterator(*'y'*)))

# Построение Координатной плоскости облака образов

fig, ax = plt.subplots(figsize=(10, 6), num=*'Обучающие множества'*)

# Чтобы перпендикуляры были перпендикулярными

ax.set\_aspect(*'equal'*, adjustable=*'box'*)

# Удаление верхней и правой границ

ax.spines[*'top'*].set\_visible(False)

ax.spines[*'left'*].set\_visible(False)

ax.spines[*'right'*].set\_visible(False)

# Добавление основных линий сетки

ax.grid(color=*'grey'*, linestyle=*'-'*, linewidth=0.25, alpha=0.5)

# Образы

ax.scatter(features\_x1, features\_y1, color=*"#8C7298"*)

ax.scatter(features\_x2, features\_y2, color=*"#be542ccc"*)

# Линия соединяющие центры облаков

lM = mlines.Line2D([cloud1.x[*'M'*], cloud2.x[*'M'*]], [cloud1.y[*'M'*], cloud2.y[*'M'*]], color=*"#000"*, linestyle=*"--"*, marker=*"x"*)

ax.add\_line(lM)

ax.annotate(*f'({cloud1.x["M"]},\n {cloud1.y["M"]}):{cloud1.klass}'*,

(cloud1.x[*"M"*], cloud1.y[*"M"*]),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

color=*'blue'*, backgroundcolor=*"#eae1e196"*)

ax.annotate(*f'({cloud2.x["M"]},\n {cloud2.y["M"]}):{cloud2.klass}'*,

(cloud2.x[*"M"*], cloud2.y[*"M"*]),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

color=*'blue'*, backgroundcolor=*"#eae1e196"*)

# Сравнитель

comparator = NNCloudComparator(cloud1, cloud2)

# Координаты середины отрезка

mid\_point = comparator.mid\_image

mid\_len = comparator.mid\_len

ax.plot(mid\_point.x, mid\_point.y, color=*"red"*, marker=*'o'*)

ax.annotate(*f'({mid\_point.x},\n {mid\_point.y})'*,

(mid\_point.x, mid\_point.y),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

color=*'blue'*, backgroundcolor=*"#eae1e196"*)

# / Координаты середины отрезка

# Координаты точка отрезка соединяющего середину и перпендикуляр

normal\_point = comparator.get\_normal\_image\_r2\_main(*'x'*, *'y'*)

lnorm = mlines.Line2D([mid\_point.x, normal\_point.x], [mid\_point.y, normal\_point.y], color=*"green"*, linestyle=*"-"*, marker=*"x"*, linewidth=0.8, )

ax.add\_line(lnorm)

# / Координаты точка отрезка соединяющего середину и перпендикуляр

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

# Program Body

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

print(*f'\nОбучающие множества\nN1={N1}, N2={N2}\nMx1={Mx1}, My1={My1}, Dx1={Dx1}, Dy1={Dy1}\nMx2={Mx2}, My2={My2}, Dx2={Dx2}, Dy2={Dy2}'*)

plt.title(*f'Обучающие множества\nN1={N1}, N2={N2}\nMx1={Mx1}, My1={My1}, Dx1={Dx1}, Dy1={Dy1}\nMx2={Mx2}, My2={My2}, Dx2={Dx2}, Dy2={Dy2}'*)

plt.show()

# Получим минимальные и максимальные значения x, y

x1\_m, y1\_m, x2\_m, y2\_m, x\_min, x\_max, y\_min, y\_max = comparator.get\_x\_y\_min\_max\_and\_m(*'x'*, *'y'*)

d\_x = (x\_max - x\_min)

x\_mid = (x\_max + x\_min) / 2

d\_y = (y\_max - y\_min)

y\_mid = (y\_max + y\_min) / 2

Dx = (abs(d\_x) / 2)\*\*2

Dy = (abs(d\_y) / 2)\*\*2

mid\_D = (Dx + Dy) / 2

mid\_s = math.sqrt(mid\_D)

Sx = math.sqrt(Dx)

Sy = math.sqrt(Dx)

Dx\_95 = (Sx \* 3)\*\*2

Dy\_95 = (Sy \* 3)\*\*2

cloud3 = ClassNormalCloud(int((abs(d\_x) \* abs(d\_y)) // (mid\_s // 2)), x={*'M'*: x\_mid, *'D'*: Dx\_95}, y={*'M'*: y\_mid, *'D'*: Dy\_95})

cloud3.fill\_cloud\_Rn\_dimension()

def **uniform\_filter**(im):

if im.x > x\_max + Sx:

return False

elif im.x < x\_min - Sx:

return False

if im.y < y\_min - Sy:

return False

elif im.y > y\_max + Sy:

return False

return True

cloud3.\_images = list(filter(uniform\_filter, cloud3.\_images)) # в области 3сигм больше всего значений. отсекаем аномалии

# Образы равномерно по плоскости

features\_x3 = list(itertools.chain(cloud3.get\_feature\_iterator(*'x'*)))

features\_y3 = list(itertools.chain(cloud3.get\_feature\_iterator(*'y'*)))

# Построение Координатной плоскости облака образов

fig, ax = plt.subplots(figsize=(10, 6), num=*'Равномерно распределенные образы'*)

# Чтобы перпендикуляры были перпендикулярными

ax.set\_aspect(*'equal'*, adjustable=*'box'*)

# Удаление верхней и правой границ

ax.spines[*'top'*].set\_visible(False)

ax.spines[*'left'*].set\_visible(False)

ax.spines[*'right'*].set\_visible(False)

# Добавление основных линий сетки

ax.grid(color=*'grey'*, linestyle=*'-'*, linewidth=0.25, alpha=0.5)

ax.scatter(features\_x3, features\_y3, color=*"#0b4b48"*)

plt.title(*f'Равномерно распределенные образы\n'*)

plt.show()

# Обучим сеть и распознаем облако

comparator.nn\_fit()

comparator.nn\_get\_recognite\_images(cloud3)

# Образы классифицированные

features\_x3 = list(itertools.chain(cloud3.get\_feature\_iterator(*'x'*)))

features\_y3 = list(itertools.chain(cloud3.get\_feature\_iterator(*'y'*)))

# Построение Координатной плоскости облака образов

fig, ax = plt.subplots(figsize=(10, 6), num=*'Классифицированные образы'*)

# Чтобы перпендикуляры были перпендикулярными

ax.set\_aspect(*'equal'*, adjustable=*'box'*)

# Удаление верхней и правой границ

ax.spines[*'top'*].set\_visible(False)

ax.spines[*'left'*].set\_visible(False)

ax.spines[*'right'*].set\_visible(False)

# Добавление основных линий сетки

ax.grid(color=*'grey'*, linestyle=*'-'*, linewidth=0.25, alpha=0.5)

# Обучающие выборки

ax.add\_line(mlines.Line2D([cloud1.x[*'M'*], cloud2.x[*'M'*]], [cloud1.y[*'M'*], cloud2.y[*'M'*]], color=*"#000"*, linestyle=*"--"*, marker=*"x"*))

ax.annotate(*f'({cloud1.x["M"]},\n {cloud1.y["M"]}):{cloud1.klass}'*,

(cloud1.x[*"M"*], cloud1.y[*"M"*]),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

color=*'blue'*, backgroundcolor=*"#eae1e196"*)

ax.annotate(*f'({cloud2.x["M"]},\n {cloud2.y["M"]}):{cloud2.klass}'*,

(cloud2.x[*"M"*], cloud2.y[*"M"*]),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

color=*'blue'*, backgroundcolor=*"#eae1e196"*)

ax.plot(mid\_point.x, mid\_point.y, color=*"red"*, marker=*'o'*)

ax.annotate(*f'({mid\_point.x},\n {mid\_point.y})'*,

(mid\_point.x, mid\_point.y),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

color=*'blue'*, backgroundcolor=*"#eae1e196"*)

normal\_point = comparator.get\_normal\_image\_r2\_main(*'x'*, *'y'*)

ax.add\_line(mlines.Line2D([mid\_point.x, normal\_point.x], [mid\_point.y, normal\_point.y], color=*"green"*, linestyle=*"-"*, marker=*"x"*, linewidth=0.8, ))

# / Обучающие выборки

for image\_ in cloud3.\_images:

if image\_.klass == cloud1.klass:

color = *'red'*

elif image\_.klass == cloud2.klass:

color = *'blue'*

else:

color = *'silver'*

ax.scatter(image\_.x, image\_.y, color=color)

# Тестирование точки. Подпись

for testpoint in [random.choice(cloud3.\_images[:5]),

random.choice(cloud3.\_images[-5:])]:

testklass = testpoint.klass

xtest = round(testpoint.x, 2)

ytest = round(testpoint.y, 2)

ax.add\_line(

mlines.Line2D(

[mid\_point.x, testpoint.x],

[mid\_point.y, testpoint.y],

color=*"purple"*,

linestyle=*'dotted'*,

linewidth=0.8,

marker=*"x"*)

)

ax.annotate(*f'({xtest},\n{ytest}),\nкласс {testklass}'*,

(testpoint.x, testpoint.y),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

color=*'#83017b'*, backgroundcolor=*"#cea7a7db"*,

)

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

# / Program Body

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

plt.title(*f'Классифицированные образы\n(Обучающие множества: N1={N1}, N2={N2}\nMx1={Mx1}, My1={My1}, Dx1={Dx1}, Dy1={Dy1}\nMx2={Mx2}, My2={My2}, Dx2={Dx2}, Dy2={Dy2})'*)

plt.show()

sys.exit()

Листинг 1. main.py

Вначале сгенерированы обучающие множества по аналогии с ЛР№ 2:

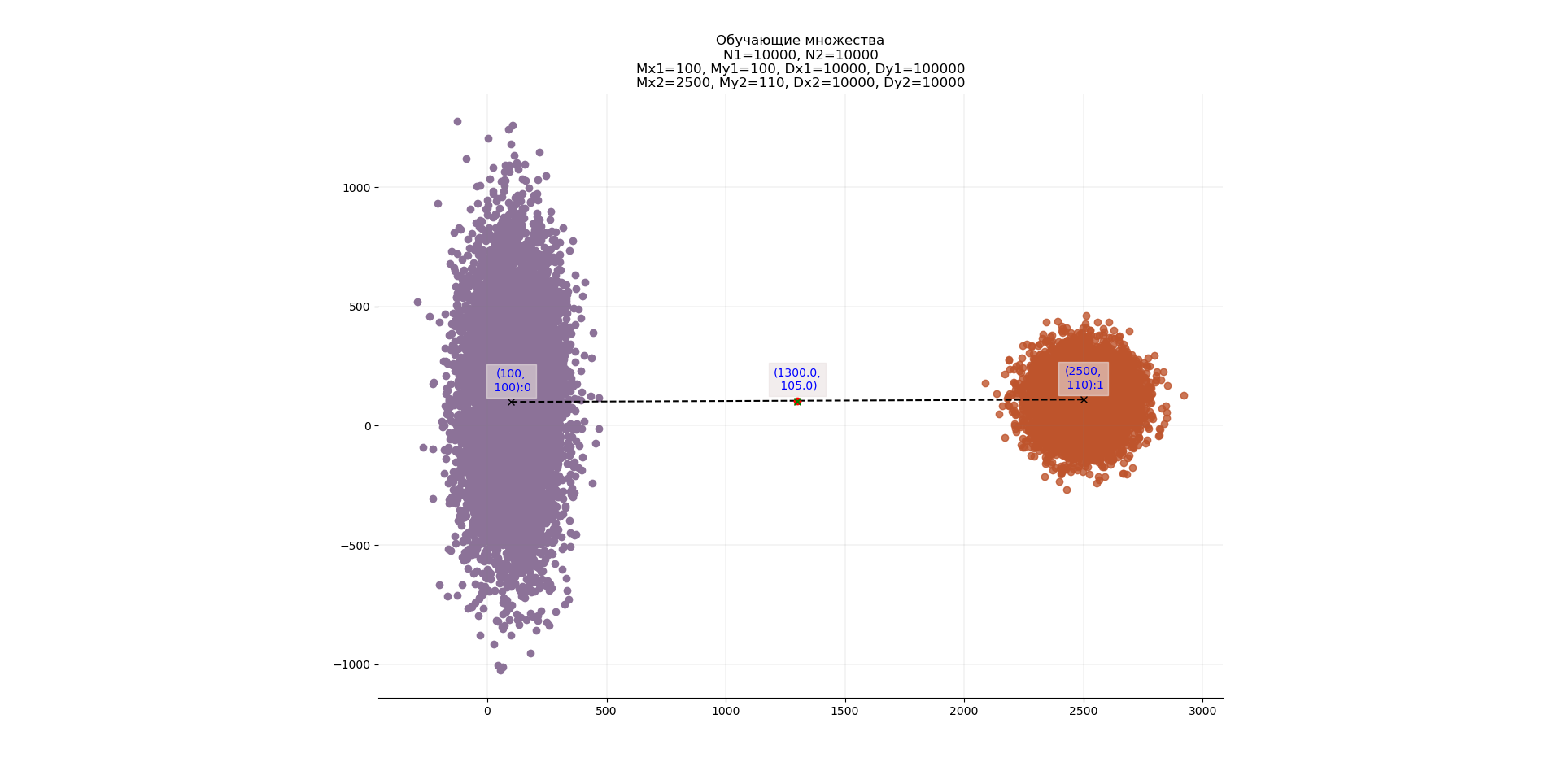


Рис 2.

Далее сгенерированы равномерно распределенные точки которые предстоит классифицировать:

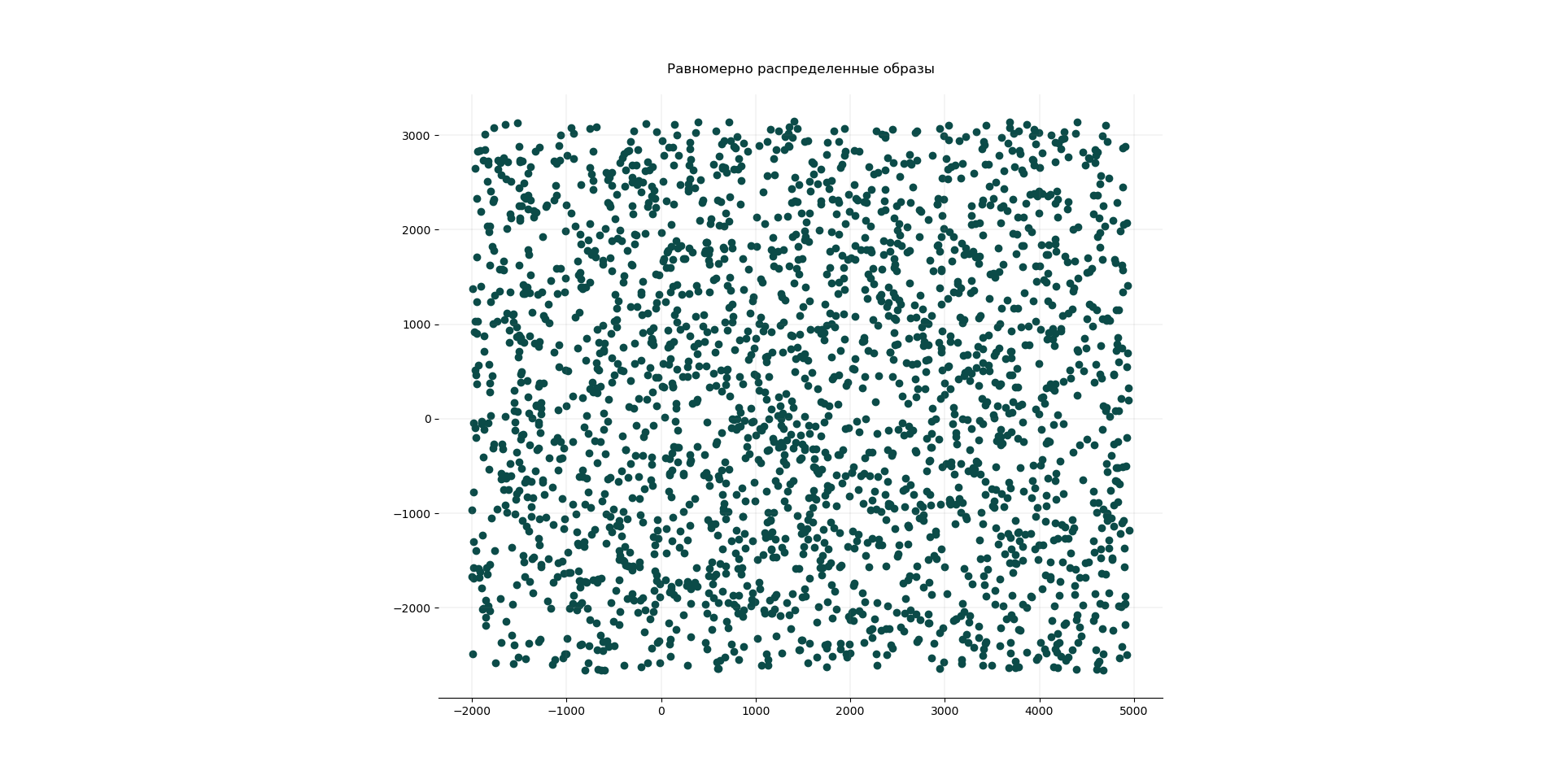


Рис 3.

После обучения сети по образам из рис 2 сеть классифицирует точки рисунка 3:

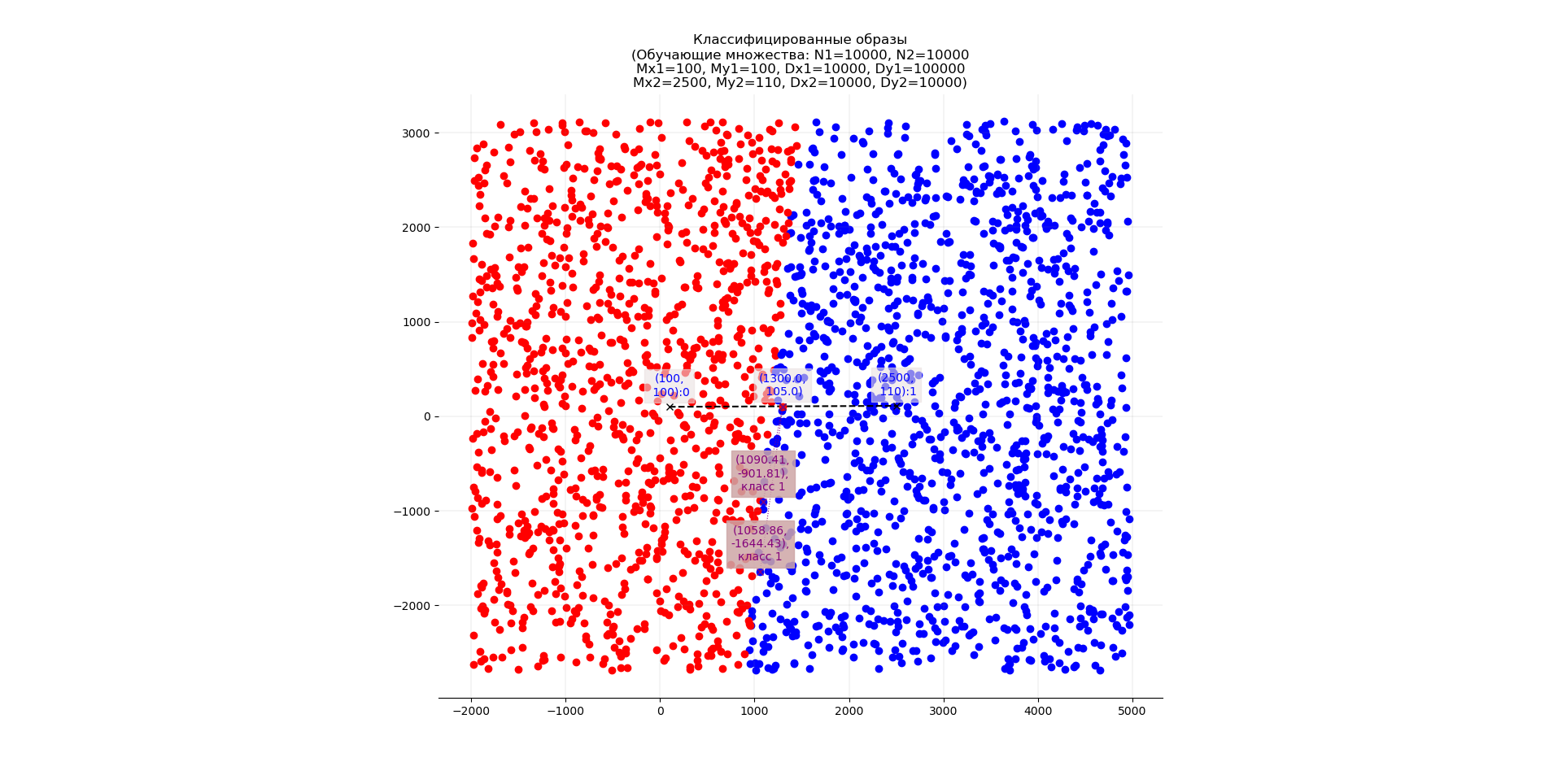


Рис 4.

# Вывод

В ходе лабораторной работы был изучена Р-модель распознавания и разработана методика классификации точек на основе ИНС.

Материалы доступны на https://github.com/PaiNt-git/study.arch\_ai/blob/main/src/lab4/main.py