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

Факультет компьютерных и фундаментальных наук

Кафедра ПОВТАС

**Лабораторная работа №2**

по дисциплине: «Архитектура систем ИИ»

на тему: «Точки (Р-модель распознавания)»  
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.

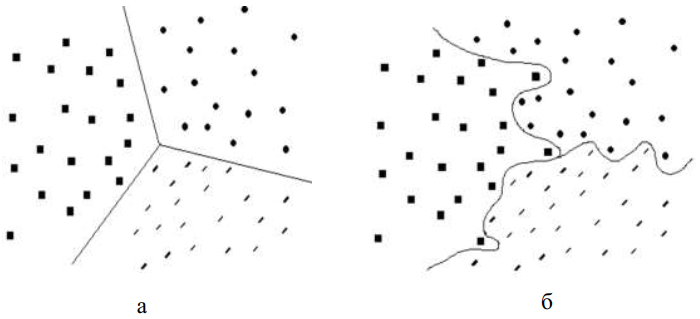


Рис. Пример разделения объектов 3-х классов

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.

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.

**CloudComparator** - «сравнитель» множеств, служебный класс имплементирующий различные операции между двумя множествами. В классе присутствуют метод, позволяющие посчитать евклидовое расстояние между математическими ожиданиями двух множеств, которое на графике иллюстрирует точки «центры» множеств, также класс содержит метод **get\_probability\_midlane\_points()**, который выдает набор точек образующих один из вариантов линии разделения «множеств» основанный на минимизации вероятности появления точки относительно каждого из множеств, получается границы минимальной вероятности (на линии соединяющей M множества) находится среднее между этими точками пересечения, класс для своей работы использует метод **pdf\_Rn\_dimension()** класса **ClassNormalCloud** , который представляет функцию расчета «плотности вероятности» нормального распределения (probability density function - PDF), в дальнейшем базовую линию разделения оптимизирует метод **get\_optimized\_sep\_line\_points()**  который вращает и смещает линию с определенным шагом – тестируя на заданных образцах принадлежность к определенному множеству с помощью метода определения угла относительно точки O (точки пересечения первой базовой линии разделения и линии соединяющей множества)

class **Image**:

*"""*

*Образ с произвольным набором признаков*

*"""*

def **\_\_init\_\_**(*self*, \*\*features):

*"""*

*features: распаковка набора признаков со значениями, значения должные быть действительными числами*

*"""*

for key, val in features.items():

if not isinstance(val, (int, float, ndarray)):

raise ValueError(*'Значения признаков должны быть действительными числами'*)

setattr(*self*, key, val)

*self*.\_dimensionality = len(features)

*self*.\_features\_names = features.keys()

*@property*

def **dimensionality**(*self*):

return *self*.\_dimensionality

*@property*

def **features\_names**(*self*):

return *self*.\_features\_names

Листинг 1. Класс Image (main.py)

class **ClassNormalCloud**:

*"""*

*Множество образов в пространстве признаков*

*"""*

def **\_\_init\_\_**(*self*, N, klass=None, \*\*Md\_ft):

*"""*

**:param** *N: размер множества*

**:param** *klass: Класс образа если задан*

*Md\_ft: распаковка параметров для нормального распределения признаков*

*Md\_ft['x'] =*

*'M': float, # математическое ожидание признака x*

*'D': float, # дисперсия случайной величины признака x*

*'cov\_lambda': None, # Функция (lambda ft\_i, ft\_j: 0) для вычисления Ковариация между i-признаком (данным) и j-признаком (где j=i+1)*

*}*

*Md\_ft['y'] =*

*'M': float, # математическое ожидание признака y*

*'D': float, # дисперсия случайной величины признака y*

*'cov\_lambda': None, # Функция (lambda ft\_i, ft\_j: 0) для вычисления Ковариация между i-признаком (данным) и j-признаком (где j=i+1)*

*}*

*...*

*Md\_ft['n'] =*

*'M': float, # математическое ожидание признака n*

*'D': float, # дисперсия случайной величины признака n*

*'cov\_lambda': None, # Функция (lambda ft\_i, ft\_j: 0) для вычисления Ковариация между i-признаком (данным) и j-признаком (где j=i+1)*

*}*

*"""*

*self*.\_features\_names = []

*self*.\_dimensionality = 0

*self*.\_size = N

*self*.\_images = []

*self*.\_default\_cov = []

*self*.klass = klass

for key, val in Md\_ft.items():

if not isinstance(key, (str, int)):

raise ValueError(*'Наименование признака должно быть строкой или натуральным числом'*)

if not isinstance(val, dict) or (*'M'* not in val) or (*'D'* not in val):

raise ValueError(*f'Параметры облака образа признака {key} должны быть словарем, содержащим как минимум ключи M и D'*)

# все норм

*self*.\_features\_names.append(key)

*self*.\_dimensionality += 1

setattr(*self*, key, {\*\*val})

for i, fnamei in enumerate(*self*.features\_names):

cov\_i\_row = []

fsetti = getattr(*self*, fnamei)

for j, fnamej in enumerate(*self*.features\_names):

fsettj = getattr(*self*, fnamej)

# Диагональ матрицы всегда дисперсия

if i == j and fnamei == fnamej:

cov\_i\_row.append(fsettj[*'D'*])

# Ковариация между i-признаком и j-признаком

else:

cov\_lambda = fsetti.get(*'cov\_lambda'*, None)

if cov\_lambda and hasattr(cov\_lambda, *'\_\_call\_\_'*):

cov\_i\_row.append(cov\_lambda(fnamei, fnamej))

else:

cov\_i\_row.append(0)

*self*.\_default\_cov.append(cov\_i\_row)

*@property*

def **dimensionality**(*self*):

return *self*.\_dimensionality

*@property*

def **features\_names**(*self*):

return *self*.\_features\_names

*@property*

def **size**(*self*):

return *self*.\_size

def **fill\_cloud\_Rn\_dimension**(*self*):

*"""*

*Заполнение множества по нормальному распределению исходя из размерности облака*

*"""*

del *self*.\_images

*self*.\_images = []

true\_dispersion = None

features\_Ms = []

for key in *self*.features\_names:

fsett = getattr(*self*, key)

if true\_dispersion is None:

true\_dispersion = fsett[*'D'*]

features\_Ms.append(fsett[*'M'*])

mean = features\_Ms

cov = *self*.\_default\_cov

\*features\_arrays, = np.random.multivariate\_normal(mean, cov, *self*.size).T

for features in itertools.zip\_longest(\*features\_arrays):

ftu = {k: v for k, v in itertools.zip\_longest(*self*.features\_names, features)}

*self*.\_images.append(Image(klass=*self*.klass, \*\*ftu))

def **pdf\_Rn\_dimension**(*self*, x: Image):

*"""*

*Пло́тность вероя́тности (probability density function - PDF)*

*Вернуть значение вероятности точки по ее признакам*

**:param** *x:*

*"""*

if x.dimensionality != *self*.dimensionality:

raise ValueError(*"Размерность образа и облака не соотносятся"*)

sigma = np.matrix(*self*.\_default\_cov)

mu = np.array([getattr(*self*, f)[*'M'*] for f in *self*.features\_names])

size = x.dimensionality

if size == len(mu) and (size, size) == sigma.shape:

features\_value = np.array([getattr(x, f) for f in *self*.features\_names])

det = np.linalg.det(sigma) # Детерминант

if det == 0:

raise ValueError(*"Ковариационная матрица не может быть сингулярной"*)

norm\_const = 1.0 / (math.pow((2 \* np.pi), float(size) / 2) \* math.pow(det, 1.0 / 2))

x\_mu = np.matrix(features\_value - mu)

inv = sigma.I

result = math.pow(math.e, -0.5 \* (x\_mu \* inv \* x\_mu.T))

return norm\_const \* result

else:

raise ValueError(*"Размерность образа и ковариационной матрицы не соотносятся"*)

pass

def **get\_feature\_iterator**(*self*, feature\_name):

for im in *self*.\_images:

yield getattr(im, feature\_name)

def **get\_feature\_list**(*self*, feature\_name):

return list(*self*.get\_feature\_iterator(feature\_name))

Листинг 2. Класс ClassNormalCloud (main.py)

class **CloudComparator**:

*"""*

*Сравнитель облак образов*

*"""*

cloud1 = None

cloud2 = None

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

*self*.cloud1 = cloud1

*self*.cloud2 = cloud2

if *self*.cloud1.dimensionality != *self*.cloud2.dimensionality:

raise ValueError(*'Размерность облаков образов не равна'*)

if *self*.cloud1.features\_names != *self*.cloud2.features\_names:

raise ValueError(*'Признаки облаков образов не совпадают'*)

*self*.\_features\_names = [x for x in *self*.cloud1.features\_names]

*self*.\_dimensionality = *self*.cloud1.dimensionality

*@property*

def **dimensionality**(*self*):

return *self*.\_dimensionality

*@property*

def **features\_names**(*self*):

return *self*.\_features\_names

*@staticmethod*

def **get\_between\_point\_len**(image1: Image, image2: Image):

*"""*

*Евклидоваое расстояние между точками*

**:param** *image1:*

**:param** *image2:*

*"""*

if image1.dimensionality != image2.dimensionality:

raise ValueError(*'Размерность образов не равна'*)

if image1.features\_names != image2.features\_names:

raise ValueError(*'Признаки образов не совпадают'*)

quads = []

for feature in image1.features\_names:

quads.append(math.pow((getattr(image2, feature) - getattr(image1, feature)), 2))

return math.sqrt(sum(quads))

*@memoize\_method*

def **get\_x\_y\_min\_max\_and\_m**(*self*, x, y):

*"""*

*Вычилить максмумы один раз*

**:param** *x:*

**:param** *y:*

*"""*

# Мат. ожидания множеств

x1\_m = getattr(*self*.cloud1, x)[*'M'*]

y1\_m = getattr(*self*.cloud1, y)[*'M'*]

x2\_m = getattr(*self*.cloud2, x)[*'M'*]

y2\_m = getattr(*self*.cloud2, y)[*'M'*]

# Вычислим минимальные и максимальные x

x\_min = min(itertools.chain(*self*.cloud1.get\_feature\_iterator(x), *self*.cloud2.get\_feature\_iterator(x)))

x\_min = x\_min - x\_min \* 0.1

x\_max = max(itertools.chain(*self*.cloud1.get\_feature\_iterator(x), *self*.cloud2.get\_feature\_iterator(x)))

x\_max = x\_max + x\_max \* 0.1

y\_min = min(itertools.chain(*self*.cloud1.get\_feature\_iterator(y), *self*.cloud2.get\_feature\_iterator(y)))

y\_min = y\_min - y\_min \* 0.1

y\_max = max(itertools.chain(*self*.cloud1.get\_feature\_iterator(y), *self*.cloud2.get\_feature\_iterator(y)))

y\_max = y\_max + y\_max \* 0.1

return x1\_m, y1\_m, x2\_m, y2\_m, x\_min, x\_max, y\_min, y\_max

*@property*

*@memoize\_method*

def **mid\_image**(*self*) -> Image:

*"""*

*Точка на середине отрезка соединяющего М множество*

*"""*

coords = {}

for feature in *self*.features\_names:

mid\_feature = ((getattr(*self*.cloud2, feature)[*'M'*] + getattr(*self*.cloud1, feature)[*'M'*]) / 2)

coords[feature] = mid\_feature

return Image(\*\*coords)

*@property*

*@memoize\_method*

def **mid\_len**(*self*):

*"""*

*Длина серединного отрезка*

*"""*

coords = {f: getattr(*self*.cloud1, f)[*'M'*] for f in *self*.features\_names}

MImage = Image(\*\*coords)

ll = *self*.get\_between\_point\_len(MImage, *self*.mid\_image)

return ll

def **get\_normal\_image\_r2\_main**(*self*, feature\_name1, feature\_name2, znak=*'+'*, ) -> Image:

*"""*

*Координаты нормали к отрезку соединяющему множества*

**:param** *feature\_name1: какой признак взять за x*

**:param** *feature\_name2: какой признак взять за y*

**:param** *znak:*

*"""*

mid\_image = *self*.mid\_image

# Обрежем размерность до R2

fe\_r2 = [feature\_name1, feature\_name2]

tgnorm\_r2 = np.round((getattr(*self*.cloud2, fe\_r2[1])[*'M'*] - getattr(*self*.cloud1, fe\_r2[1])[*'M'*]) / (getattr(*self*.cloud2, fe\_r2[0])[*'M'*] - getattr(*self*.cloud1, fe\_r2[0])[*'M'*]), 4)

norm\_r2\_len = (*self*.mid\_len \* tgnorm\_r2)

if not norm\_r2\_len:

norm\_r2\_len = *self*.get\_between\_point\_len(

Image(\*\*{feature\_name1: getattr(*self*.cloud1, feature\_name1)[*'M'*], feature\_name2: getattr(*self*.cloud1, feature\_name2)[*'M'*]}),

Image(\*\*{feature\_name1: getattr(*self*.cloud2, feature\_name1)[*'M'*], feature\_name2: getattr(*self*.cloud2, feature\_name2)[*'M'*]}),

) / 2

coords = {}

featquad = sum([math.pow((getattr(*self*.cloud2, f)[*'M'*] - getattr(*self*.cloud1, f)[*'M'*]), 2) for f in fe\_r2])

for i, feature in enumerate(fe\_r2):

if znak == *'+'*:

if i == 0:

coords[feature] = getattr(mid\_image, fe\_r2[0]) + norm\_r2\_len \* ((getattr(*self*.cloud2, fe\_r2[1])[*'M'*] - getattr(*self*.cloud1, fe\_r2[1])[*'M'*]) / math.sqrt(featquad))

else:

coords[feature] = getattr(mid\_image, fe\_r2[1]) - norm\_r2\_len \* ((getattr(*self*.cloud2, fe\_r2[0])[*'M'*] - getattr(*self*.cloud1, fe\_r2[0])[*'M'*]) / math.sqrt(featquad))

else:

if i == 0:

coords[feature] = getattr(mid\_image, fe\_r2[0]) - norm\_r2\_len \* ((getattr(*self*.cloud2, fe\_r2[1])[*'M'*] - getattr(*self*.cloud1, fe\_r2[1])[*'M'*]) / math.sqrt(featquad))

else:

coords[feature] = getattr(mid\_image, fe\_r2[1]) + norm\_r2\_len \* ((getattr(*self*.cloud2, fe\_r2[0])[*'M'*] - getattr(*self*.cloud1, fe\_r2[0])[*'M'*]) / math.sqrt(featquad))

# Заполним координаты недолстающих признаков нулями

not\_has\_features = [x for x in *self*.features\_names if x not in fe\_r2]

coords.update((k, 0) for k in not\_has\_features)

return Image(\*\*coords)

...

def **get\_probability\_midlane\_points**(*self*, margin\_of\_error=0.0005, ax=None, fi=0, offset\_from\_O=0):

*"""*

*Нарисовать линию разделения между облаками на основе минимизации вероятности pdf*

**:param** *cloud\_num: относительно какого облака рисовать*

**:param** *margin\_of\_error:*

**:param** *ax:*

*"""*

cloud1\_features\_value = [getattr(*self*.cloud1, f) for f in *self*.cloud1.features\_names]

cloud2\_features\_value = [getattr(*self*.cloud2, f) for f in *self*.cloud2.features\_names]

M1max = step\_accuracy1 = max([x[*'M'*] for x in cloud1\_features\_value])

M2max = step\_accuracy2 = max([x[*'M'*] for x in cloud2\_features\_value])

D1max = max(x[*'D'*] for x in cloud1\_features\_value)

margin\_of\_error1 = margin\_of\_error / D1max

D2max = max(x[*'D'*] for x in cloud2\_features\_value)

margin\_of\_error2 = margin\_of\_error / D2max

# Приведение гиперпространства к сумме двухмерных

images = []

line\_equations = {}

cloud\_plus\_points = []

cloud\_minus\_points = []

step\_x = 0.5

step\_y = 0.5

# Вычислим минимальные и максимальные x

x\_min = 0

x\_max = 0

y\_min = 0

y\_max = 0

for cloud\_num in (1, 2):

current\_cloud = *self*.cloud1 if cloud\_num == 1 else *self*.cloud2

current\_margin = margin\_of\_error1 if cloud\_num == 1 else margin\_of\_error2

current\_step\_accuracy = step\_accuracy1 if cloud\_num == 1 else step\_accuracy2

for feature\_r2 in itertools.combinations(current\_cloud.features\_names, 2):

x, y = feature\_r2

x\_m = getattr(current\_cloud, x)[*'M'*]

y\_m = getattr(current\_cloud, y)[*'M'*]

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

not\_has\_features = [f for f in current\_cloud.features\_names if f not in (x, y)]

# шаги изменений для алгоритма

step\_x = x\_m / current\_step\_accuracy

step\_y = y\_m / current\_step\_accuracy

step\_len = min((step\_x, step\_y))

# определяем наибольшие точки относительно шага по линии

# Уравнение линии соединения

k\_base = k((x1\_m, y1\_m), (x2\_m, y2\_m))

b\_base = b((x1\_m, y1\_m), (x2\_m, y2\_m))

# Координаты середины отрезка

mid\_point = *self*.mid\_image

# Координаты точка отрезка соединяющего середину и перпендикуляр

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

# Уравнение перепендикуляра к середине

k\_normal = k((getattr(mid\_point, x), getattr(mid\_point, y)), (getattr(normal\_point, x), getattr(normal\_point, y)))

b\_normal = b((getattr(mid\_point, x), getattr(mid\_point, y)), (getattr(normal\_point, x), getattr(normal\_point, y)))

if b\_normal in (-inf, inf) or isnan(b\_normal):

coords\_1 = {

x: (x1\_m + x2\_m) / 2,

y: y\_max,

}

coords\_1.update((f, getattr(current\_cloud, f)[*'M'*]) for f in not\_has\_features)

coords\_2 = {

x: (x1\_m + x2\_m) / 2,

y: y\_min,

}

coords\_2.update((f, getattr(current\_cloud, f)[*'M'*]) for f in not\_has\_features)

if not (x, y, 0) in line\_equations:

\_equations = {

*'func\_def'*: *f'''{y} = {k\_normal}\*x + {b\_normal}'''*,

*'k'*: k\_normal,

*'b'*: b\_normal,

*f'{y}'*: lambda xx: pretty\_line\_y(b\_normal, y\_min, y\_max, (k\_normal \* xx + b\_top\_plus)),

*'center\_point'*: *self*.mid\_image,

}

line\_equations[(x, y, 0)] = \_equations

...

images.append(Image(\*\*coords\_1))

images.append(Image(\*\*coords\_2))

return images, line\_equations

else:

# Функция смещения для прямой

def **b\_frompoint\_func**(xx, yy): return (yy \* getattr(normal\_point, x) - xx \* ((getattr(normal\_point, x) - xx) \* k\_normal + yy)) / (getattr(normal\_point, x) - xx)

# Перебор точек по линии (+)

\_iter\_counter = 0

current\_x = x\_m

current\_y = y\_m

coords\_top\_plus = {

x: current\_x,

y: current\_y,

}

coords\_top\_plus.update((f, getattr(current\_cloud, f)[*'M'*]) for f in not\_has\_features)

while abs(current\_cloud.pdf\_Rn\_dimension(Image(\*\*coords\_top\_plus))) > current\_margin:

current\_x = current\_x + step\_len \* \_iter\_counter

current\_y = pretty\_line\_y(b\_base, y\_min, y\_max, (k\_base \* current\_x + b\_base))

coords\_top\_plus[x] = pretty\_line\_x(b\_base, current\_x, current\_x)

coords\_top\_plus[y] = current\_y

\_iter\_counter += 1

# Точка перпендикуляра

# Перебор точек по противолинии (-)

\_iter\_counter = 0

current\_x = x\_m

current\_y = y\_m

coords\_top\_minus = {

x: current\_x,

y: current\_y,

}

coords\_top\_minus.update((f, getattr(current\_cloud, f)[*'M'*]) for f in not\_has\_features)

while abs(current\_cloud.pdf\_Rn\_dimension(Image(\*\*coords\_top\_minus))) > current\_margin:

current\_x = current\_x - step\_len \* \_iter\_counter

current\_y = pretty\_line\_y(b\_base, y\_min, y\_max, (k\_base \* current\_x + b\_base))

coords\_top\_minus[x] = pretty\_line\_x(b\_base, current\_x, current\_x)

coords\_top\_minus[y] = current\_y

\_iter\_counter += 1

if not (x, y, cloud\_num) in line\_equations:

b\_top\_plus = b\_frompoint\_func(coords\_top\_plus[x], coords\_top\_plus[y])

b\_top\_minus = b\_frompoint\_func(coords\_top\_minus[x], coords\_top\_minus[y])

\_equations = {

*'func\_def'*: *f'''{y}\_minus = {k\_normal}\*x + {b\_top\_plus};\n{y}\_plus = {k\_normal}\*x + {b\_top\_minus}'''*,

*'k'*: k\_normal,

*'b\_minus'*: b\_top\_plus,

*'b\_plus'*: b\_top\_minus,

*f'{y}\_plus'*: lambda xx: pretty\_line\_y(b\_top\_plus, y\_min, y\_max, (k\_normal \* xx + b\_top\_plus)),

*f'{y}\_minus'*: lambda xx: pretty\_line\_y(b\_top\_minus, y\_min, y\_max, (k\_normal \* xx + b\_top\_minus)),

*'b\_offset\_func'*: b\_frompoint\_func,

}

line\_equations[(x, y, cloud\_num)] = \_equations

cloud\_plus\_points.append((cloud\_num, *'plus'*, Image(\*\*coords\_top\_plus)))

cloud\_minus\_points.append((cloud\_num, *'minus'*, Image(\*\*coords\_top\_minus)))

# Вычислим комбинациями минимальное расстояние

minlen = None

cloud\_lens = {}

for points in itertools.permutations(cloud\_plus\_points + cloud\_minus\_points, 2):

# Если это координаты одинаковых облак

if points[0][0] == points[1][0]:

continue

# Если это минусы с минусами

if points[0][1] == points[1][1]:

continue

ml = *self*.get\_between\_point\_len(points[0][2], points[1][2])

cloud\_lens[str(ml)] = (points[0][2], points[1][2])

if minlen is None or ml < minlen:

minlen = ml

# Координаты середины между пересечениями

center\_coords = {}

for feature in *self*.features\_names:

mid\_feature = ((getattr(cloud\_lens[str(minlen)][1], feature) + getattr(cloud\_lens[str(minlen)][0], feature)) / 2)

center\_coords[feature] = mid\_feature

midinm = Image(\*\*center\_coords)

# Вычислим точку перпендикуляра к середине между границами распределения

endnormal\_coords = []

for feature\_r2 in itertools.combinations(*self*.cloud1.features\_names, 2):

x, y = feature\_r2

x\_m = getattr(current\_cloud, x)[*'M'*]

y\_m = getattr(current\_cloud, y)[*'M'*]

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

not\_has\_features = [f for f in current\_cloud.features\_names if f not in (x, y)]

\_equations1 = line\_equations[(x, y, 1)]

\_equations2 = line\_equations[(x, y, 2)]

k\_endnormal = (\_equations1[*'k'*] + \_equations2[*'k'*]) / 2

b\_endnormal = (\_equations1[*'b\_offset\_func'*](getattr(midinm, x), getattr(midinm, y)) + \_equations2[*'b\_offset\_func'*](getattr(midinm, x), getattr(midinm, y))) / 2

if not (x, y, 0) in line\_equations:

\_equations = {

*'func\_def'*: *f'''{y} = {k\_endnormal}\*x + {b\_endnormal}'''*,

*'k'*: k\_endnormal,

*'b'*: b\_endnormal,

*f'{y}'*: lambda xx: pretty\_line\_y(b\_endnormal, y\_min, y\_max, (k\_endnormal \* xx + b\_endnormal)),

*'center\_point'*: midinm,

}

line\_equations[(x, y, 0)] = \_equations

x1 = pretty\_line\_x(b\_endnormal, getattr(midinm, x), x\_min)

y1 = pretty\_line\_y(b\_endnormal, y\_min, y\_max, (k\_endnormal \* x1 + b\_endnormal))

x2 = pretty\_line\_x(b\_endnormal, getattr(midinm, x), x\_max)

y2 = pretty\_line\_y(b\_endnormal, y\_min, y\_max, (k\_endnormal \* x2 + b\_endnormal))

endnormal\_coords.append({

x: x1,

y: y1,

})

endnormal\_coords.append({

x: x2,

y: y2,

})

# Вращение прямой относительно точки пересечения

for feature\_r2 in itertools.combinations(*self*.cloud1.features\_names, 2):

x, y = feature\_r2

k\_rotate, b\_rotate, left\_x, left\_y, right\_x, right\_y = rotate\_line\_equation(line\_equations[(x, y, 0)][*'k'*],

line\_equations[(x, y, 0)][*'b'*],

(getattr(line\_equations[(x, y, 0)][*'center\_point'*], x), getattr(line\_equations[(x, y, 0)][*'center\_point'*], y)), # o\_point

(endnormal\_coords[0][x], endnormal\_coords[0][y]), # r\_point

x\_min, x\_max, y\_min, y\_max, step\_x

)(fi)

# Если задано смещение offset\_from\_O

if offset\_from\_O:

k\_rotate, b\_rotate, left\_x, left\_y, right\_x, right\_y = offset\_line\_equation\_and\_points(k\_rotate,

b\_rotate,

(getattr(line\_equations[(x, y, 0)][*'center\_point'*], x), getattr(line\_equations[(x, y, 0)][*'center\_point'*], y)), # o\_point

(endnormal\_coords[0][x], endnormal\_coords[0][y]), # r\_point

x\_min, x\_max, y\_min, y\_max)(offset\_from\_O)

...

images.append(Image(\*\*{

x: left\_x,

y: left\_y,

}))

images.append(Image(\*\*{

x: right\_x,

y: right\_y,

}))

return images, line\_equations

def **\_test\_point\_sepline\_alpha**(*self*, testpoint: Image, k\_sep\_line, b\_sep\_line, left\_cloud\_num=None):

*"""*

*С помощьюф функции math.atan2 и уравнению по паралельному переносу и смещению системы координат*

*вычислим (угол) квадрант в который попадает линия соединяющая тестируемую точку со смещенным началом координат*

*http://www.pm298.ru/dekart.php*

**:param** *testpoint:*

**:param** *k\_sep\_line:*

**:param** *b\_sep\_line:*

**:param** *left\_cloud\_num:*

*"""*

left\_cloud\_num\_ = left\_cloud\_num

ret\_alfas = {}

for feature\_r2 in itertools.combinations(*self*.cloud1.features\_names, 2):

x, y = feature\_r2

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

if not left\_cloud\_num:

left\_cloud\_num\_ = 1 if x1\_m <= x2\_m else 2

not\_has\_features = [f for f in *self*.cloud1.features\_names if f not in (x, y)]

# Уравнение линии соединения

k\_base = k((x1\_m, y1\_m), (x2\_m, y2\_m))

b\_base = b((x1\_m, y1\_m), (x2\_m, y2\_m))

O\_x = pretty\_line\_x(b\_sep\_line, (x1\_m + x2\_m) / 2, (b\_sep\_line - b\_base) / (k\_base - k\_sep\_line))

O\_y = pretty\_line\_y(b\_base, y\_min, y\_max, (k\_base \* O\_x + b\_base))

test\_x, tesy\_y = getattr(testpoint, x), getattr(testpoint, y)

# Угол наклона линии соединения базовой

alfa\_base = math.atan(k\_base)

# Координаты середины отрезка

mid\_point = *self*.mid\_image

# Координаты точка отрезка соединяющего середину и перпендикуляр

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

k\_normal = k((getattr(mid\_point, x), getattr(mid\_point, y)), (getattr(normal\_point, x), getattr(normal\_point, y)))

### b\_normal = b((getattr(mid\_point, x), getattr(mid\_point, y)), (getattr(normal\_point, x), getattr(normal\_point, y)))

# Угол наклона нормали

alfa\_normal = math.atan(k\_normal)

# Угол наклона линии разделения

alfa\_sep\_O = math.atan(k\_sep\_line)

# Угол смещения

delta\_fi = math.pi - alfa\_normal - (math.pi - alfa\_sep\_O)

### delta\_fi\_grad = math.degrees(math.pi - alfa\_normal - (math.pi - alfa\_sep\_O))

# Уравнение x\_off, y\_off для координат относительно линии соединяющей

def **x\_off**(xx, yy): return ((xx - O\_x) \* math.cos(alfa\_base) + (yy - O\_y) \* math.sin(alfa\_base))

def **y\_off**(xx, yy): return (-(xx - O\_x) \* math.sin(alfa\_base) + (yy - O\_y) \* math.cos(alfa\_base))

# Уравнение x\_off, y\_off для координат относительно линии соединяющей + delta\_fi

def **x\_off\_delta\_fi**(xx, yy): return ((xx - O\_x) \* math.cos(alfa\_base + delta\_fi) + (yy - O\_y) \* math.sin(alfa\_base + delta\_fi))

def **y\_off\_delta\_fi**(xx, yy): return (-(xx - O\_x) \* math.sin(alfa\_base + delta\_fi) + (yy - O\_y) \* math.cos(alfa\_base + delta\_fi))

# Угол смещения относительно перепендикуляра к разделяющей

### alfa\_off\_normal = math.degrees(math.atan2(y\_off(test\_x, tesy\_y), x\_off(test\_x, tesy\_y)))

# Угол смещения относительно перепендикуляра к разделяющей + delta\_fi

alfa\_off\_normal\_delta\_fi = math.degrees(math.atan2(y\_off\_delta\_fi(test\_x, tesy\_y), x\_off\_delta\_fi(test\_x, tesy\_y)))

if not (x, y) in ret\_alfas:

\_equations = {

*f'{x}\_off'*: x\_off,

*f'{y}\_off'*: y\_off,

*'alfa'*: alfa\_off\_normal\_delta\_fi,

}

ret\_alfas[(x, y)] = \_equations

return ret\_alfas

def **classify\_image**(*self*, testpoint: Image, k\_sep\_line, b\_sep\_line, left\_cloud\_num=None):

*"""*

*Вернуть класс облака которому принадлежит точка*

**:param** *testpoint:*

**:param** *k\_sep\_line:*

**:param** *b\_sep\_line:*

**:param** *left\_cloud\_num:*

*"""*

left\_cloud\_num\_ = left\_cloud\_num

alfas = *self*.\_test\_point\_sepline\_alpha(testpoint, k\_sep\_line=k\_sep\_line, b\_sep\_line=b\_sep\_line, left\_cloud\_num=left\_cloud\_num)

verdicts = []

for feature\_r2 in itertools.combinations(*self*.cloud1.features\_names, 2):

x, y = feature\_r2

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

if not left\_cloud\_num:

left\_cloud\_num\_ = 1 if x1\_m <= x2\_m else 2

alfa = round(alfas[(x, y)][*'alfa'*], 2)

if (180 > alfa > 90) or (-180 < alfa < -90):

if left\_cloud\_num\_ == 1:

verdicts.append(1)

else:

verdicts.append(2)

elif (90 > alfa > 0) or (-90 < alfa < 0):

if left\_cloud\_num\_ == 1:

verdicts.append(2)

else:

verdicts.append(1)

else:

verdicts.append(-1)

verdicts.append(-1)

cloud1\_count = verdicts.count(1)

cloud2\_count = verdicts.count(2)

cloud\_undefined\_count = verdicts.count(-1)

if cloud1\_count > cloud2\_count and (not cloud\_undefined\_count):

return *self*.cloud1.klass or 1

elif cloud1\_count < cloud2\_count and (not cloud\_undefined\_count):

return *self*.cloud2.klass or 2

else:

return None

def **get\_optimized\_sep\_line\_points**(*self*, training1: list, training2: list, k\_line, b\_line, r\_point\_im: Image, steps=20, left\_cloud\_num=None):

*"""*

*Оптимизировать линию, смещая и поворачивая*

**:param** *training1: итерируемое образов для проверки*

**:param** *training2: итерируемое образов для проверки*

**:param** *k\_line:*

**:param** *b\_line:*

**:param** *r\_point\_im: точка на прямой не лежащая на линии пересечения ее с соединяющей линией*

**:param** *steps:*

**:param** *left\_cloud\_num:*

*"""*

left\_cloud\_num\_ = left\_cloud\_num

ret\_k = 0

ret\_b = 0

ret\_offset = 0

images = []

line\_equations = {}

good\_points = defaultdict(dict)

for feature\_r2 in itertools.combinations(*self*.cloud1.features\_names, 2):

x, y = feature\_r2

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

if not left\_cloud\_num:

left\_cloud\_num\_ = 1 if x1\_m <= x2\_m else 2

# Уравнение линии соединения

k\_base = k((x1\_m, y1\_m), (x2\_m, y2\_m))

b\_base = b((x1\_m, y1\_m), (x2\_m, y2\_m))

r\_x = getattr(r\_point\_im, x)

r\_y = getattr(r\_point\_im, y)

odd\_stnum = 0

even\_stnum = 0

stvalue = int(abs(getattr(*self*.cloud1, x)[*'M'*] - getattr(*self*.cloud2, x)[*'M'*]) / 20)

offsets = []

for step in range(0, steps + 1):

iseven = (step % 2 == 0)

if iseven:

even\_stnum += 1

offsets.append(-even\_stnum \* stvalue)

else:

odd\_stnum += 1

offsets.append(odd\_stnum \* stvalue)

left\_x, left\_y, right\_x, right\_y = x\_min, y\_min, x\_max, y\_max

k\_rot\_off = k\_line

b\_rot\_off = b\_line

for offset in offsets:

O\_x = pretty\_line\_x(b\_rot\_off, (x1\_m + x2\_m) / 2, (b\_rot\_off - b\_base) / (k\_base - k\_rot\_off))

O\_y = pretty\_line\_y(b\_base, y\_min, y\_max, (k\_base \* O\_x + b\_base))

k\_rot\_off, b\_rot\_off, left\_x, left\_y, right\_x, right\_y = offset\_line\_equation\_and\_points(k\_rot\_off,

b\_rot\_off,

(O\_x, O\_y), # o\_point

(r\_x, r\_y), # r\_point

x\_min, x\_max, y\_min, y\_max

)(offset)

k\_offset\_r = round(k\_rot\_off, 4)

b\_offset\_r = round(b\_rot\_off, 4)

if (k\_offset\_r, b\_offset\_r, offset) not in good\_points[(x, y)]:

good\_points[(x, y)][(k\_offset\_r, b\_offset\_r, offset)] = 0

for testpoint in itertools.chain(training1, training2):

testklass = comparator.classify\_image(testpoint, k\_sep\_line=k\_rot\_off, b\_sep\_line=b\_rot\_off, left\_cloud\_num=left\_cloud\_num\_)

if testklass == testpoint.klass:

good\_points[(x, y)][(k\_offset\_r, b\_offset\_r, offset)] = good\_points[(x, y)][(k\_offset\_r, b\_offset\_r, offset)] + 1

else:

#good\_points[(x, y)][(k\_offset\_r, b\_offset\_r, offset)] = good\_points[(x, y)][(k\_offset\_r, b\_offset\_r, offset)] - 1

pass

linegp = good\_points[(x, y)][(k\_offset\_r, b\_offset\_r, offset)]

# print(f'Линия y = {k\_offset\_r} \* x + {b\_offset\_r}, Хороших точек: {linegp}')

for fi\_ in range(20, 91, 20):

O\_x = pretty\_line\_x(b\_rot\_off, (x1\_m + x2\_m) / 2, (b\_rot\_off - b\_base) / (k\_base - k\_rot\_off))

O\_y = pretty\_line\_y(b\_base, y\_min, y\_max, (k\_base \* O\_x + b\_base))

k\_rot\_off\_, b\_rot\_off\_, left\_x, left\_y, right\_x, right\_y = rotate\_line\_equation(k\_rot\_off,

b\_rot\_off,

(O\_x, O\_y), # o\_point

(left\_x, left\_y), # r\_point

x\_min, x\_max, y\_min, y\_max

)(fi\_)

k\_rotate\_r = round(k\_rot\_off\_, 4)

b\_rotate\_r = round(b\_rot\_off\_, 4)

if (k\_rotate\_r, b\_rotate\_r, offset) not in good\_points[(x, y)]:

good\_points[(x, y)][(k\_rotate\_r, b\_rotate\_r, offset)] = 0

for testpoint in itertools.chain(training1, training2):

testklass = comparator.classify\_image(testpoint, k\_sep\_line=k\_rot\_off\_, b\_sep\_line=b\_rot\_off\_, left\_cloud\_num=left\_cloud\_num\_)

if testklass == testpoint.klass:

good\_points[(x, y)][(k\_rotate\_r, b\_rotate\_r, offset)] = good\_points[(x, y)][(k\_rotate\_r, b\_rotate\_r, offset)] + 1

else:

#good\_points[(x, y)][(k\_rotate\_r, b\_rotate\_r, offset)] = good\_points[(x, y)][(k\_rotate\_r, b\_rotate\_r, offset)] - 1

pass

linegp = good\_points[(x, y)][(k\_rotate\_r, b\_rotate\_r, offset)]

# print(f'Линия y = {k\_rotate\_r} \* x + {b\_rotate\_r}, Хороших точек: {linegp}')

pass

tmax = max(good\_points[(x, y)].values())

for key, val in good\_points[(x, y)].items():

if val == tmax:

ret\_k = key[0]

ret\_b = key[1]

ret\_offset = key[2]

break

O\_x = pretty\_line\_x(ret\_b, (x1\_m + x2\_m) / 2, (ret\_b - b\_base) / (k\_base - ret\_k))

O\_y = pretty\_line\_y(b\_base, y\_min, y\_max, (k\_base \* O\_x + b\_base))

if not (x, y) in line\_equations:

\_equations = {

*'func\_def'*: *f'''{y} = {ret\_k}\*x + {ret\_b}'''*,

*'k'*: ret\_k,

*'b'*: ret\_b,

*f'{y}'*: lambda xx: pretty\_line\_y(ret\_b, y\_min, y\_max, (ret\_k \* xx + ret\_b)),

*'center\_point'*: Image(\*\*{

x: O\_x,

y: O\_y,

}),

}

line\_equations[(x, y)] = \_equations

step\_x = abs((x2\_m - x1\_m) / (steps \* 2))

def **formul**(xx): return ret\_k \* xx + ret\_b

if ret\_b not in (-inf, inf) and ret\_k not in (-inf, inf):

# "Левая" часть, возрастание и убывание функций

left\_x = x\_min

left\_y = formul(left\_x)

...

opt\_features\_x1 = [

pretty\_line\_x(ret\_b, O\_x, left\_x),

pretty\_line\_x(ret\_b, O\_x, right\_x),

]

images.append(Image(\*\*{

x: opt\_features\_x1[0],

y: pretty\_line\_y(ret\_b, y\_min, y\_max, formul(opt\_features\_x1[0])),

}))

images.append(Image(\*\*{

x: opt\_features\_x1[1],

y: pretty\_line\_y(ret\_b, y\_min, y\_max, formul(opt\_features\_x1[1])),

}))

else:

if ret\_k in (-inf, inf) or isnan(ret\_k):

betta = math.radians(90)

else:

betta = math.atan(ret\_k)

lenline = y\_max - O\_y

O\_x = O\_x + ret\_offset

line\_equations[(x, y)][*f'center\_point'*] = Image(\*\*{

x: O\_x,

y: O\_y,

})

left\_x, right\_x = lenline \* math.cos(betta) + O\_x, -lenline \* math.cos(betta) + O\_x

left\_y, right\_y = lenline \* math.sin(betta) + O\_y, -lenline \* math.sin(betta) + O\_y

images.append(Image(\*\*{

x: left\_x,

y: left\_y,

}))

images.append(Image(\*\*{

x: right\_x,

y: right\_y,

}))

...

return images, line\_equations

*@staticmethod*

def **get\_feature\_iterator\_from\_images**(images, feature\_name):

for im in images:

yield getattr(im, feature\_name)

pass

Листинг 3. Класс CloudComparator (main.py)

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

...

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

cloud1 = ClassNormalCloud(N1, x={*'M'*: Mx1, *'D'*: Dx1}, y={*'M'*: My1, *'D'*: Dy1}, klass=2)

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=*'Множества образов'*)

# 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 = CloudComparator(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

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

# Разделение через минимум Пло́тности вероя́тности

# Линия перпендикулярная

sep\_points, line\_equations = comparator.get\_probability\_midlane\_points(ax=ax, fi=0)

le = line\_equations[(*'x'*, *'y'*, 0)]

sep\_features\_x1 = list(CloudComparator.get\_feature\_iterator\_from\_images(sep\_points, *'x'*))

sep\_features\_y1 = list(CloudComparator.get\_feature\_iterator\_from\_images(sep\_points, *'y'*))

ax.add\_line(

mlines.Line2D(

sep\_features\_x1,

sep\_features\_y1,

color=*"black"*,

linestyle=*'dashdot'*,

linewidth=1.5,

marker=*""*)

)

# Линия оптимизированная

opt\_sep\_points, opt\_line\_equations = comparator.get\_optimized\_sep\_line\_points(cloud1.\_images[:200], cloud2.\_images[:200],

le[*'k'*], le[*'b'*],

Image(x=le[*'x\_r\_point'*], y=le[*'y\_r\_point'*]),

steps=10)

opt\_le = opt\_line\_equations[(*'x'*, *'y'*)]

opt\_sep\_features\_x1 = list(CloudComparator.get\_feature\_iterator\_from\_images(opt\_sep\_points, *'x'*))

opt\_sep\_features\_y1 = list(CloudComparator.get\_feature\_iterator\_from\_images(opt\_sep\_points, *'y'*))

ax.add\_line(

mlines.Line2D(

opt\_sep\_features\_x1,

opt\_sep\_features\_y1,

color=*"red"*,

linewidth=2.1,

marker=*""*)

)

# Тестирование точки. Подпись угла

for testpoint in [random.choice(cloud1.\_images[:5]),

random.choice(cloud2.\_images[:5]),

random.choice(cloud1.\_images[-5:]),

random.choice(cloud2.\_images[-5:])]:

testklass = comparator.classify\_image(testpoint, k\_sep\_line=opt\_le[*'k'*], b\_sep\_line=opt\_le[*'b'*])

xtest = round(testpoint.x, 2)

ytest = round(testpoint.y, 2)

alfas = comparator.\_test\_point\_sepline\_alpha(testpoint, k\_sep\_line=opt\_le[*'k'*], b\_sep\_line=opt\_le[*'b'*])

alfa = round(alfas[(*'x'*, *'y'*)][*'alfa'*], 2)

ax.add\_line(

mlines.Line2D(

[opt\_le[*'center\_point'*].x, testpoint.x],

[opt\_le[*'center\_point'*].y, testpoint.y],

color=*"purple"*,

linestyle=*'dotted'*,

linewidth=0.8,

marker=*"x"*)

)

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

(testpoint.x, testpoint.y),

textcoords=*"offset points"*,

xytext=(0, 10),

ha=*'center'*,

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

)

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

# / Program Body

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

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

plt.title(*f'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 3. Работа с графиками if \_\_name\_\_ == "\_\_main\_\_": (main.py)

Получены результаты работы метода для различных параметров множеств:

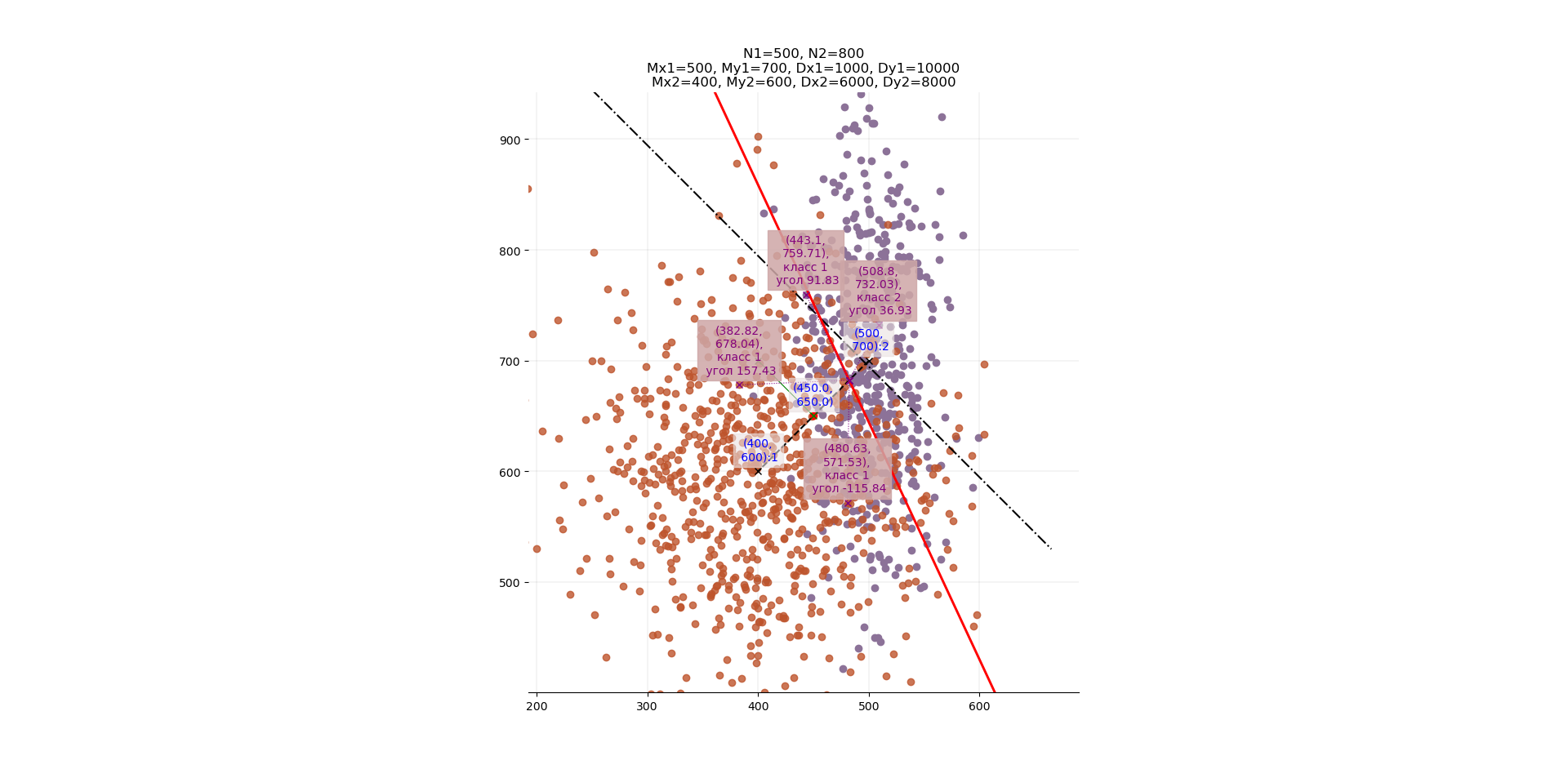


Рис 2.

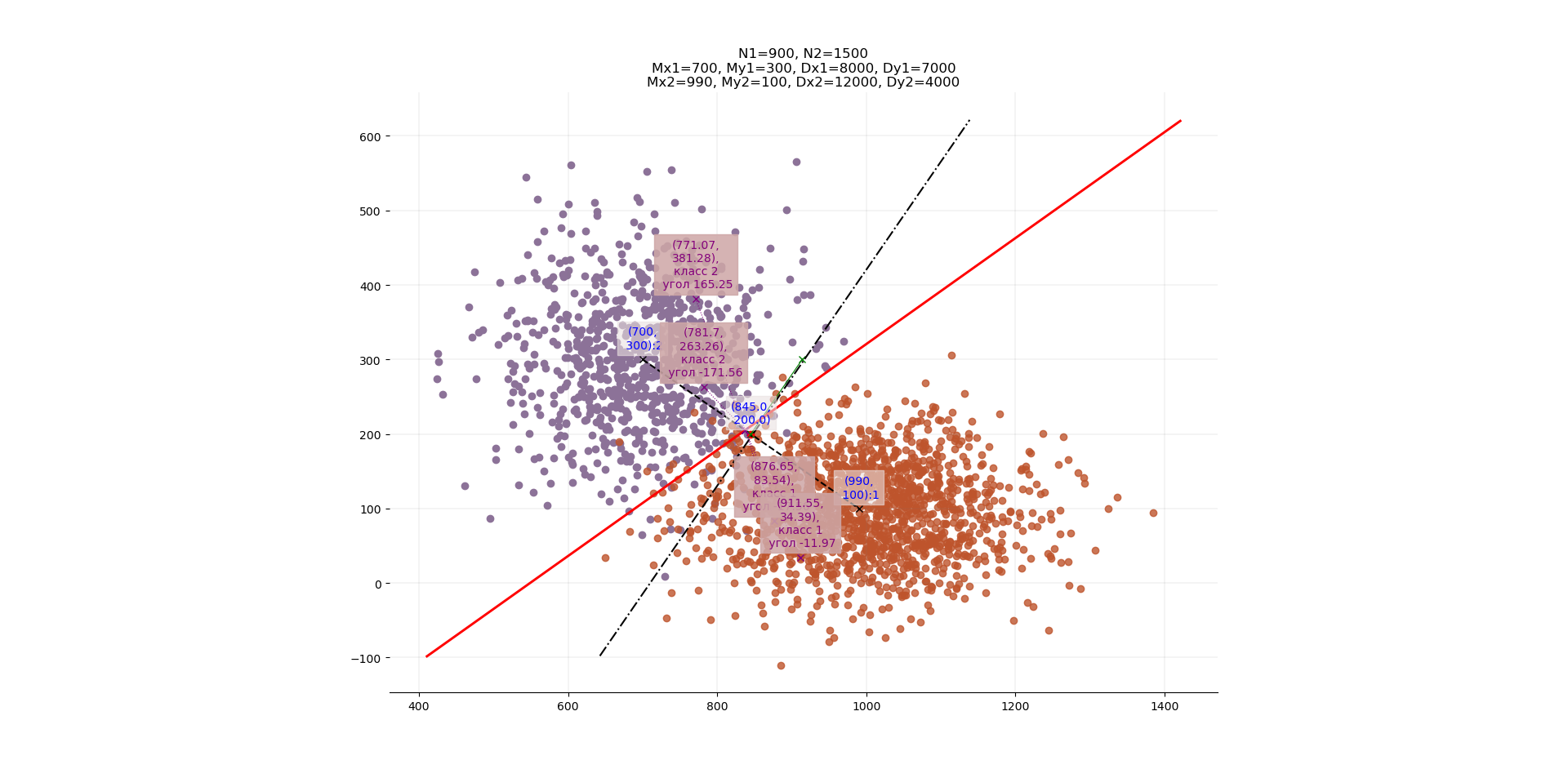


Рис 3.

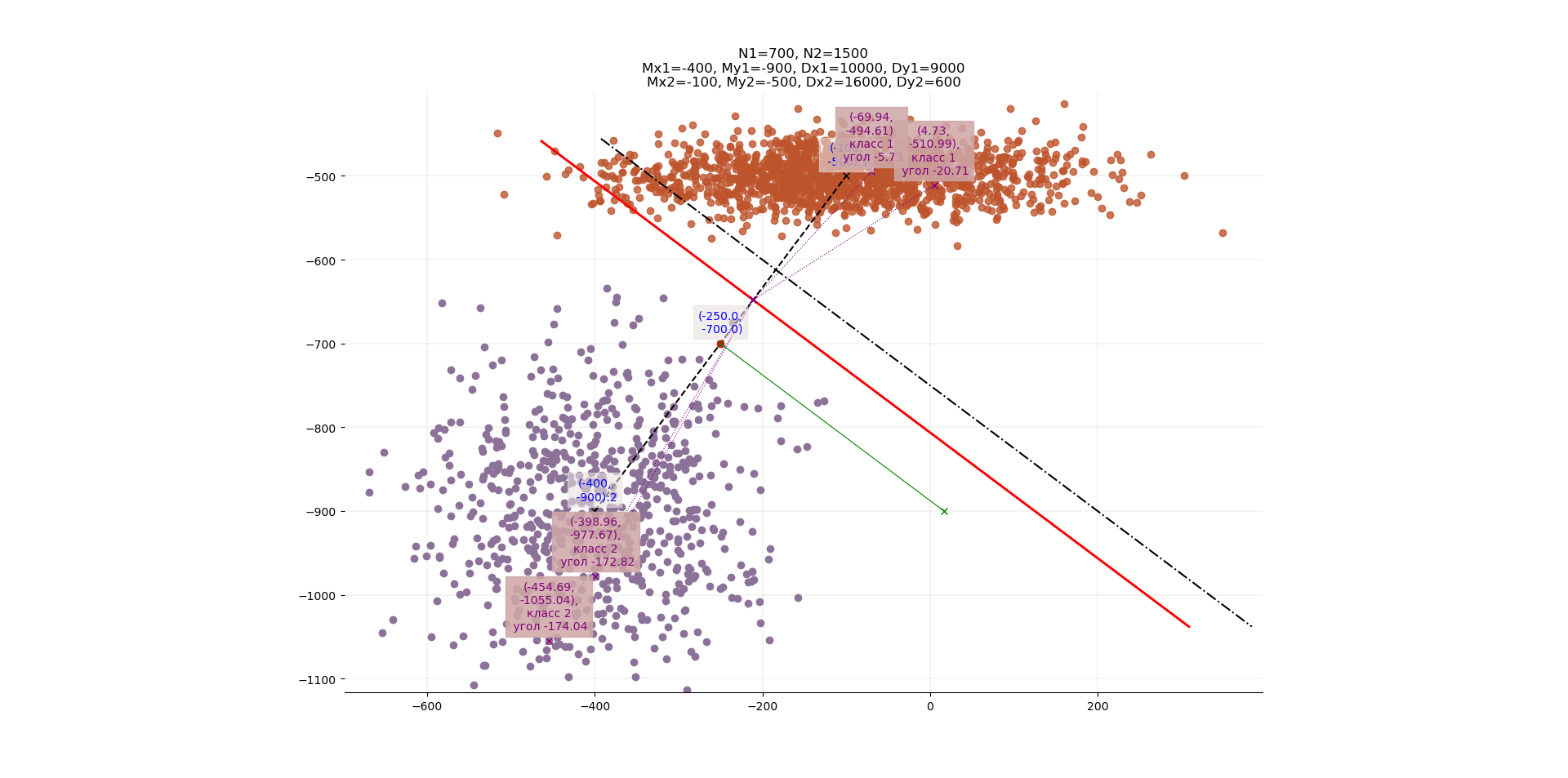


Рис 4.

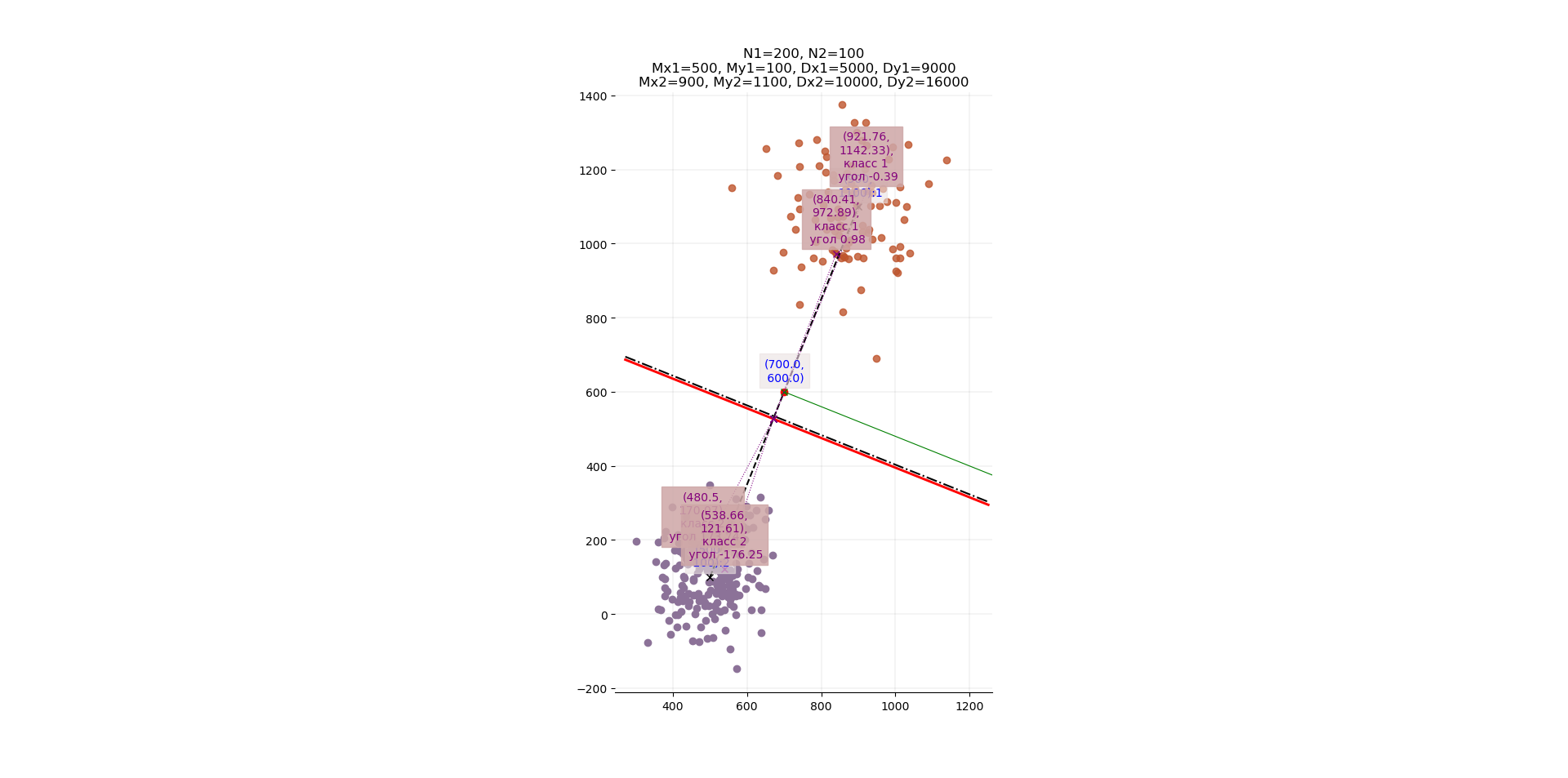


Рис 5.

# Вывод

В ходе лабораторной работы был изучена Р-модель распознавания и разработана методика построения разделяющей гиперплоскости между объектами 2 различных классов с помощью закона нормального распределения.