**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.

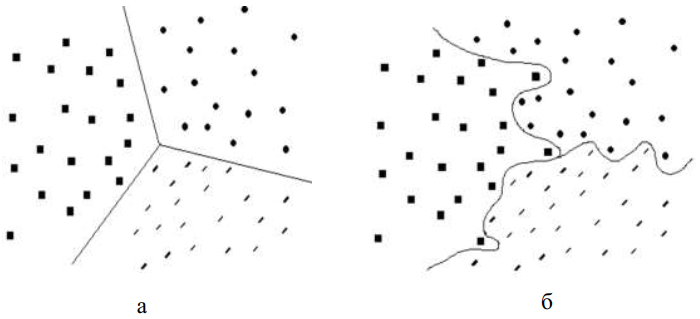


Рис. Пример разделения объектов 3-х классов

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).

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 облако образов 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» облаков, также класс содержит метод **get\_separate\_hyperplane\_points()** , который выдает набор точек образующих один из вариантов линии разделения «облак» основанный на минимизации вероятности появления точки, класс для своей работы использует метод **pdf\_Rn\_dimension()** класса **ClassNormalCloud** , который представляет функцию расчета «плотности вероятности» нормального распределения (probability density function - PDF)

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 . Класс Image (main.py)

class **ClassNormalCloud**:

*"""*

*Облако образов в пространстве признаков*

*"""*

def **\_\_init\_\_**(*self*, N, \*\*Md\_ft):

*"""*

**:param** *N: размер облака*

*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 = []

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**(*self*):

*"""*

*Обычное заполнение облака образами с \_независимым\_ (получается одномерным) нормальным распределением каждого признака*

*"""*

del *self*.\_images

*self*.\_images = []

features\_appropriate = []

for key in *self*.features\_names:

fsett = getattr(*self*, key)

sko = math.sqrt(fsett[*'D'*])

features\_appropriate.append(np.nditer(np.random.normal(fsett[*'M'*], sko, *self*.size))) # fsett['M']-матожидание величины признака sko-СКО self.size.-размер массива

for i, features in enumerate(itertools.zip\_longest(\*features\_appropriate)):

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

*self*.\_images.append(Image(\*\*ftu))

pass

pass

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

*"""*

*Заполнение облака 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'*]

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

# if fsett['D'] != true\_dispersion:

# raise ValueError('В режиме заливки "по полной размерности облака" необходимо равенство дисперсий каждого признака')

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

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(\*\*ftu))

pass

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

*"""*

*Пло́тность вероя́тности (probability density function - PDF) - scypy*

*Вернуть значение вероятности точки по ее признакам*

**:param** *x:*

*"""*

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

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

cov\_m = *self*.\_default\_cov

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

norm\_distribution = sps.multivariate\_normal(mean=mu, cov=cov\_m)

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

return norm\_distribution.pdf(np.array(features\_value))

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))

*@property*

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

*"""*

*Точка на середине отрезка соединяющего облака*

*"""*

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*

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 облака*

**: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)

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\_separate\_hyperplane\_points**(*self*, cloud\_num=1, margin\_of\_error=0.0005):

*"""*

*Нарисовать линию (окружность) разделения между облаками*

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

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

*"""*

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

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

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

# Приведение гиперпространства к сумме двухмерных

images = []

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'*]

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 / step\_accuracy1

step\_y = y\_m / step\_accuracy1

# определяем наибольшие точки сверху

\_iter\_counter = 0

current\_y = y\_m + step\_y \* \_iter\_counter

coords\_top\_y = {

x: x\_m,

y: current\_y,

}

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

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

current\_y = y\_m + step\_y \* \_iter\_counter

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

\_iter\_counter += 1

top\_y = step\_y \* \_iter\_counter

# определяем наибольшие точки справа

\_iter\_counter = 0

current\_x = x\_m + step\_x \* \_iter\_counter

coords\_top\_x = {

x: current\_x,

y: y\_m,

}

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

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

current\_x = x\_m + step\_x \* \_iter\_counter

coords\_top\_x[x] = current\_x

\_iter\_counter += 1

top\_x = step\_x \* \_iter\_counter

for al in range(0, 361, 30):

x\_ = x\_m + top\_x \* math.cos(math.radians(al))

y\_ = y\_m + top\_y \* math.sin(math.radians(al))

coords\_ = {

x: x\_,

y: y\_,

}

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

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

return images

*@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\_\_"*:

cloud1 = ClassNormalCloud(100, x={*'M'*: 500, *'D'*: 1000}, y={*'M'*: 300, *'D'*: 10000})

cloud1.fill\_cloud\_Rn\_dimension()

cloud2 = ClassNormalCloud(100, x={*'M'*: 40, *'D'*: 30000}, y={*'M'*: 100, *'D'*: 2000})

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=*'Облака 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.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.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"*)

ax.add\_line(lnorm)

# / Координаты точка отрезка соединяющего середину и перпендикуляр

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

# Program Body

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

# Разделение через минимум Пло́тности вероя́тности

sep\_plane\_images = comparator.get\_separate\_hyperplane\_points(1)

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

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

for i in range(1, len(sep\_plane\_images), 1):

ax.add\_line(

mlines.Line2D(

[sep\_plane\_images[i - 1].x, sep\_plane\_images[i].x],

[sep\_plane\_images[i - 1].y, sep\_plane\_images[i].y],

color=*"#e6188c"*,

marker=*"x"*)

)

sep\_plane\_images = comparator.get\_separate\_hyperplane\_points(2)

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

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

for i in range(1, len(sep\_plane\_images), 1):

ax.add\_line(

mlines.Line2D(

[sep\_plane\_images[i - 1].x, sep\_plane\_images[i].x],

[sep\_plane\_images[i - 1].y, sep\_plane\_images[i].y],

color=*"#44ec86"*,

marker=*"x"*)

)

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

# / Program Body

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

plt.show()

sys.exit()

Листинг 3. Работа с графиками if \_\_name\_\_ == "\_\_main\_\_": (main.py)

Получены результаты работы метода для различных параметров множеств:

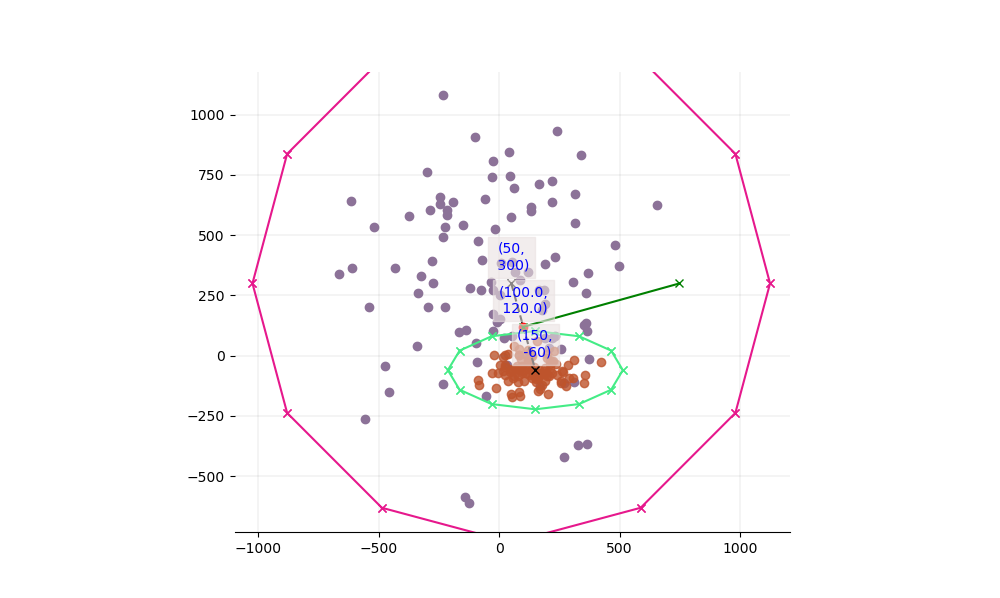


Рис 2. Cloud1: x={'M': 50, 'D': 100000}, y={'M': 300, 'D': 100000}

Cloud2: x={'M': 150, 'D': 10000}, y={'M': -60, 'D': 2000}

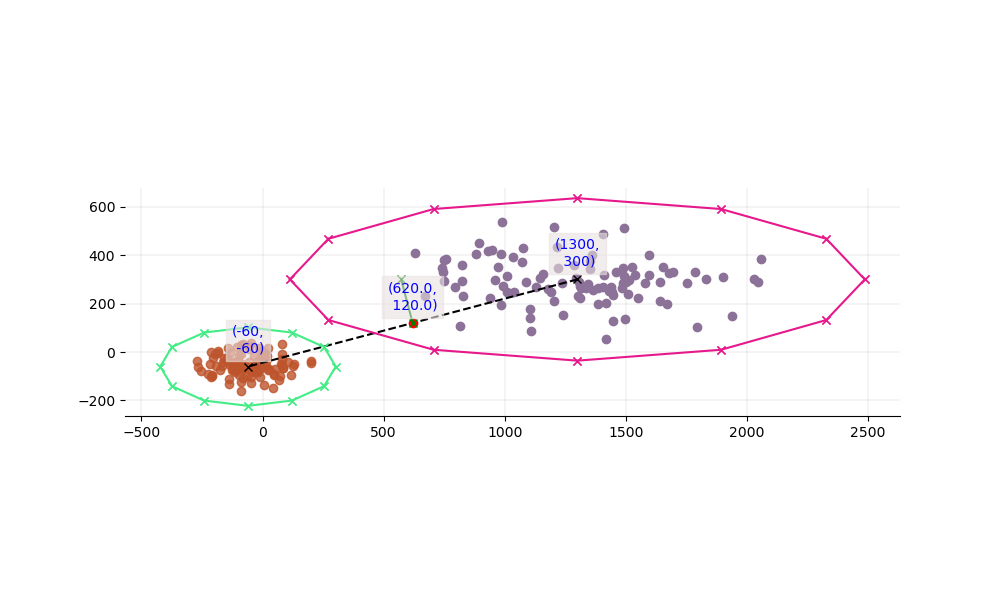


Рис 3. Cloud1: x={'M': 1300, 'D': 100000}, y={'M': 300, 'D': 8000}

Cloud2: x={'M': -60, 'D': 10000}, y={'M': -60, 'D': 2000}

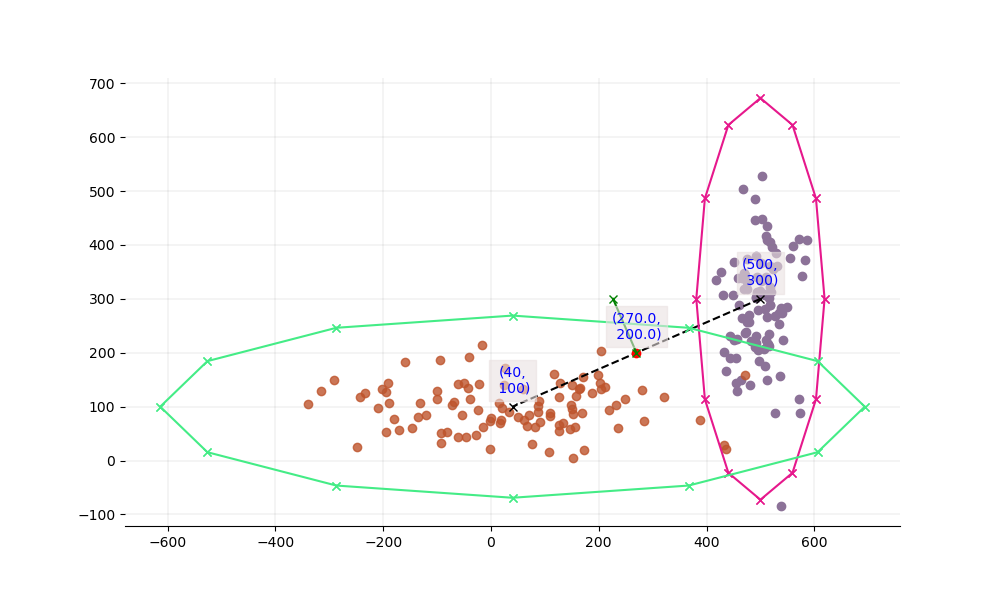


Рис 4. Cloud1: x={'M': 500, 'D': 1000}, y={'M': 300, 'D': 10000}

Cloud2: x={'M': 40, 'D': 30000}, y={'M': 100, 'D': 2000}

# 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.