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 (P-модель распознавания)»

Выполнил: студент группы ПИИ(м)-21
Забавин А.С.

Проверил: ст. преп. кафедры ПОВТАС
Тормозов В.С.

Хабаровск, 2022

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 N_1 – количество образцов из первого класса и N_2 – количество образцов из второго класса. N_1+N_2 пар чисел (x_k,y_k) для образцов из первого и второго классов.

Требуется выполнить графическую иллюстрацию P-модели.

Замечание

Точки разных классов могут задаваться пользователем произвольно или генерироваться автоматически. Для автоматического формирования наборов точек (x_k,y_k) каждого класса следует воспользоваться следующей информацией. Пусть в пространстве признаков R^2 заданы два нормальных распределения с математическими ожиданиями (Mx_1,My_1) и (Mx_2,My_2) и дисперсиями σ_1 и σ_2.

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 N_1+N_2 объектов, считаем, что исходная информация задана.

Таким образом, при разработке программы следует предусмотреть ввод пользователем величин N_1, N_2, Mx_1, My_1, Mx_2, My_2, σ_1 и σ_2.

Краткая теория

P-модель (модель разделения) характеризуется тем, что проводиться граница между классами в пространстве R^n размерности n. При построении информационного вектора $\alpha(w^*)$ исследуется положение объекта w^* 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.

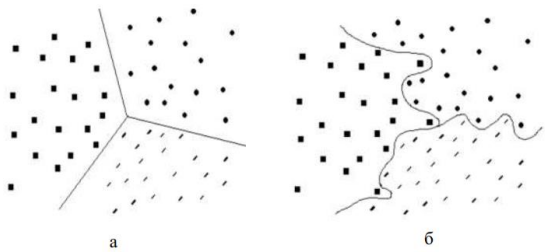

Рис. 1 Пример разделения объектов 3-х классов

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 P-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 R^n 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 действи-
тельными 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:
    """
    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" необхо-
димо 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):
        """
        Плóтность 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):
        """
        Плóтность 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):
        """
        Евклидовое 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, fea-
ture)['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))) > cur-
rent_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))) > cur-
rent_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']], col-
or="#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:

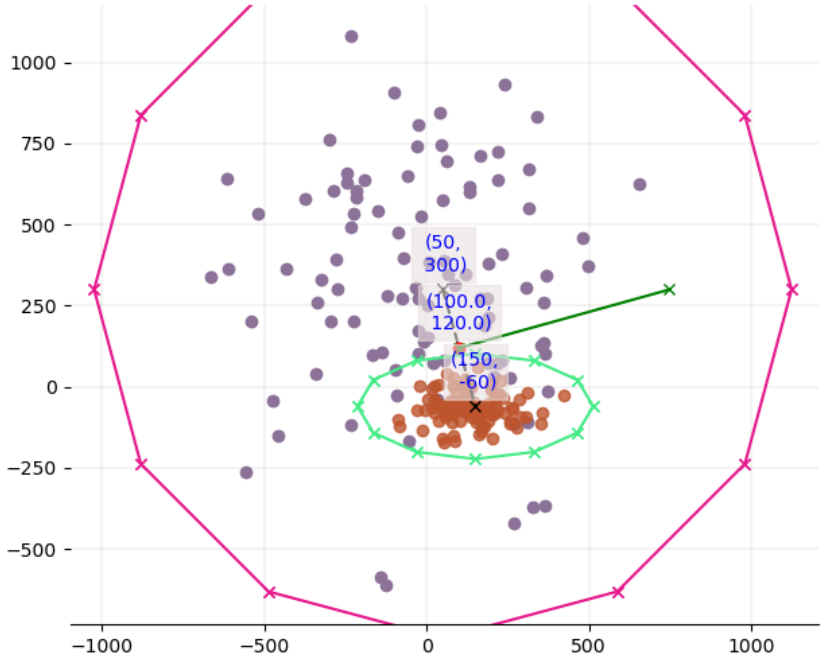

Рис 2. Cloud1: x={'M': 50, 'D': 100000}, y={'M': 300, 'D': 100000}

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

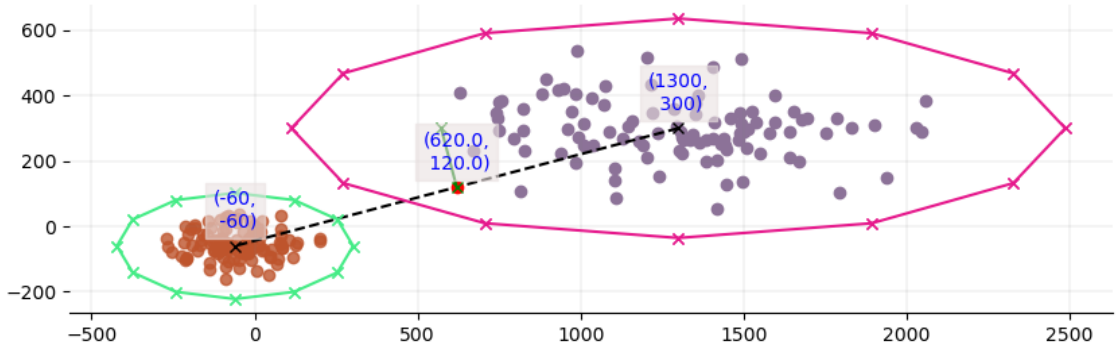

Рис 3. Cloud1: x={'M': 1300, 'D': 100000}, y={'M': 300, 'D': 8000}

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

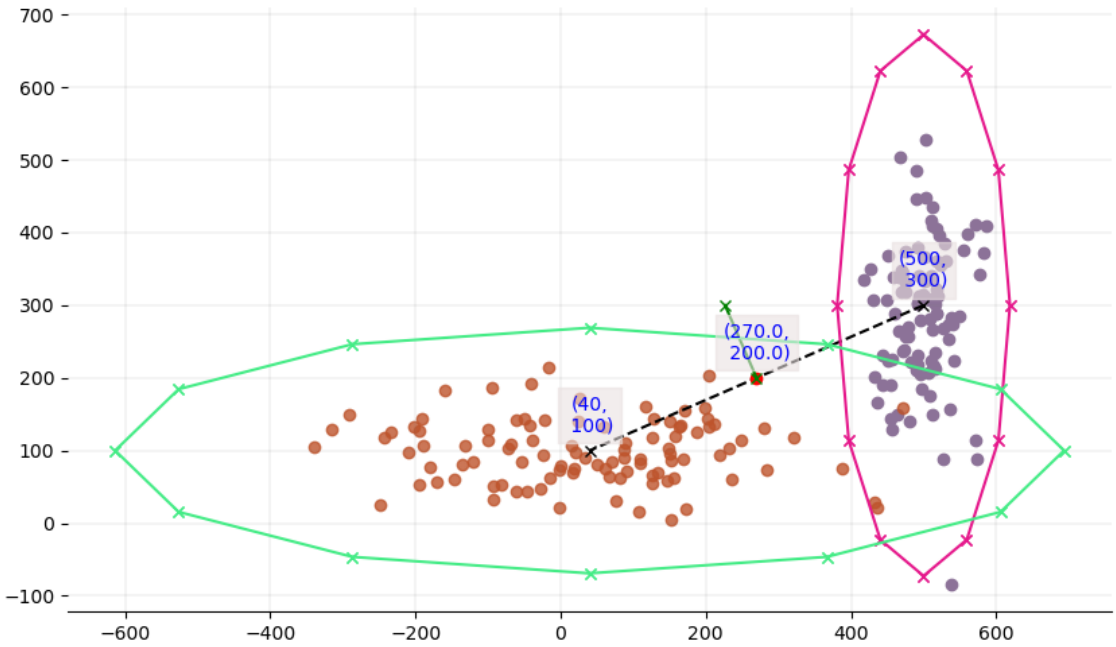

Рис 4. Cloud1: x={'M': 500, 'D': 1000}, y={'M': 300, 'D': 10000}

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.