**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: «Нейронные сети для решения задачи классификации»  
Вариант №4

Выполнил: студент группы ПИИ(м)-21

Латынцев А.В.

Проверил: преподаватель 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 с использованием методов искусственных нейронных сетей.

**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 жиданиями (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.

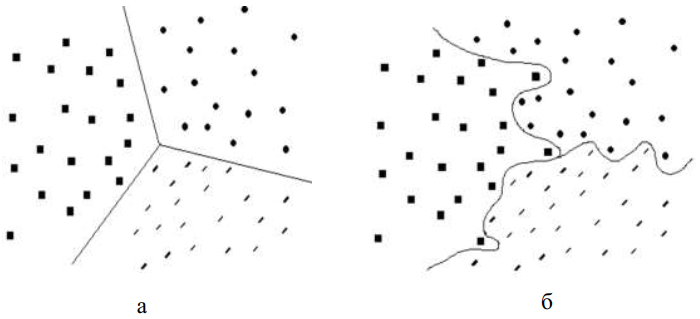


Рис. Пример разделения объектов 3-х классов

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 на языке программирования Python 3 с использованием Jupyter Notebook. Замечу, что ниже я не буду подробно расписывать те или иные решения, поскольку в листингах либо все подробно прокомментировано, либо названия методов и переменных говорят сами за себя.

Первым делом (Листинг 1) определимся с тем, что нам понадобится:

|  |
| --- |
| **import** **scipy.stats** **as** **sps** # 1.9.2  **import** **numpy** **as** **np** # 1.23.4  **import** **ipywidgets** **as** **widgets** # 8.0.2  **import** **matplotlib.pyplot** **as** **plt** # 3.6.1  **import** **matplotlib.lines** **as** **mlines**  **from** **sklearn** **import** svm # 1.1.2  **from** **sklearn.linear\_model** **import** LogisticRegression  **from** **math** **import** cos, pi, sin  # использовать системное приложение для взаимодействия с графиками  # в моем случае - TkAgg  %matplotlib  **def** **grad\_to\_rad**(grad):  **return** grad/**360**\*pi\***2** |

Листинг 1. Используемые библиотеки и функция перевода радиан в градусы

Следующий шаг – определение класса для создания объектов 2-мерного нормального распределения (см. Листинг 2).

|  |
| --- |
| **class** **norm\_distribution**:  **def** **\_\_init\_\_**(self, N: int, math\_exp: list, cov\_m: list):  """  N: кол-во образцов из первого класса  math\_exp: математическое ожидание [Mx1, My1]  cov: ковариацион. матрица распределения  [[variance\_x, 0], [0, variance\_y]]  """    self.math\_exp = math\_exp  self.cov\_m = cov\_m    # Объект нормального распределения  self.norm\_distribution = \  sps.multivariate\_normal(mean=math\_exp, cov=cov\_m)    self.points = \  self.norm\_distribution.rvs(size=N) # наши N точки    # Метод для ручного добавления точек  **def** **add\_point**(self, x, y):  self.points = list(self.points)  self.points.append(np.array([x, y]))  self.points = np.array(self.points)  # Вернуть значение функции плотности вероятности для точек  # (по факту считается очень маленький интервал)  **def** **return\_probability**(self, x, y):  **return** self.norm\_distribution.pdf(np.array([x,y]))    # Просто для удобства  **def** **show\_points**(self, count\_of\_points=**0**):  **if** count\_of\_points == **0**:  **print**("Значения выборки:**\n**", self.points[:])  **else**:  **print**(  f"Первые {count\_of\_points} значений выборки:**\n**",  self.points[:count\_of\_points]  )    # Просто для удобства  **def** **show\_3d\_plot\_of\_distribution**(self, x\_domain=**0**, y\_domain=**0**):  """  x\_domain и y\_domain  конечные координаты на 3D графике соответственно;  по умолчанию - 0, что означает - подобрать самостоятельно  """  # Предустановка масштаба, если нужно  **if** x\_domain == **0**:  self.x\_domain = (self.cov\_m[**0**][**0**] + self.math\_exp[**0**]) \* **2**  **else**:  self.x\_domain = x\_domain  **if** y\_domain == **0**:  self.y\_domain = (self.cov\_m[**1**][**1**] + self.math\_exp[**1**]) \* **2**  **else**:  self.y\_domain = y\_domain      # Создаем сетку и многомерную нормаль  x,y = \  np.linspace(-self.x\_domain, self.x\_domain,**500**), \  np.linspace(-self.y\_domain, self.y\_domain,**500**)  X,Y = np.meshgrid(x,y) # компануем в сетку  # Определяем ось вероятностей  pos = np.empty(X.shape + (**2**,))  pos[:, :, **0**] = X; pos[:, :, **1**] = Y  Z = self.norm\_distribution.pdf(pos)    # Строем 3D график  fig = plt.figure(figsize=(**10**,**10**))  self.ax = fig.add\_subplot(projection='3d')  self.ax.plot\_surface(  X, Y, Z,  cmap='viridis',linewidth=**0**  )  # Установка осей  self.ax.set\_xlabel('X axis')  self.ax.set\_ylabel('Y axis')  self.ax.set\_zlabel('Z axis')  plt.show() # показать график |

Листинг . Реализация класса norm\_distribution

Листинг 3 описывает класс **distribution\_analysis**, предназначенный для сравнения 2-х экземпляров класса **norm\_distribution**. Содержит методы для определения гиперплоскости и её отрисовки; а также для отрисовки «облаков»; линии, соединяющей их центры и просто базовых вещей (координатной сетки, подписей, масштаба и т.п.).

|  |
| --- |
| **class** **distribution\_analysis**:  **def** **\_\_init\_\_**(self, object\_1, object\_2, graph\_title: str):  """  object\_1 и object\_2 - экземпляры класса norm\_distribution  """  self.object\_1, self.object\_2 = object\_1, object\_2    # Построение Координатной плоскости облака образов  fig, self.ax = plt.subplots(  figsize=(**10**, **10**),  num=graph\_title  )  self.initialization\_of\_graph()    **def** **plot\_everything**(self, show: bool, hyperplane\_mode="svc"):  self.add\_norm\_points\_on\_graph()  self.add\_connect\_centers\_line()  self.add\_hyperplane(mode=hyperplane\_mode)    **if** show:  self.ax.plot()      **def** **initialization\_of\_graph**(self):  self.ax.set\_aspect('equal', adjustable='box')    # Удаление верхней и правой границ  self.ax.spines['top'].set\_visible(False)  self.ax.spines['left'].set\_visible(False)  self.ax.spines['right'].set\_visible(False)    # Добавление основных линий сетки  self.ax.grid(color='grey', linestyle='-', linewidth=**0.25**, alpha=**0.5**)      **def** **add\_norm\_points\_on\_graph**(self, points\_color\_1='#454FA1', points\_color\_2='#E01A2D'):  self.ax.scatter(  np.array(list(map(**lambda** value: value[**0**], self.object\_2.points))),  np.array(list(map(**lambda** value: value[**1**], self.object\_2.points))),  color=points\_color\_2  ) # x2, y2    self.ax.scatter(  np.array(list(map(**lambda** value: value[**0**], self.object\_1.points))),  np.array(list(map(**lambda** value: value[**1**], self.object\_1.points))),  color=points\_color\_1  )    **def** **add\_connect\_centers\_line**(self):  # для удобства  x1\_m, y1\_m = self.object\_1.math\_exp[**0**], self.object\_1.math\_exp[**1**]  x2\_m, y2\_m = self.object\_2.math\_exp[**0**], self.object\_2.math\_exp[**1**]    lM = mlines.Line2D(  [x1\_m, x2\_m], [y1\_m, y2\_m],  color="#000", linestyle="--", marker="x"  )  self.ax.add\_line(lM)  self.ax.annotate(f'({x1\_m}; {y1\_m})',  (x1\_m, y1\_m),  textcoords="offset points",  xytext=(**0**, **10**),  ha='center',  color='blue', backgroundcolor="#eae1e196")  self.ax.annotate(f'({x2\_m}; {y2\_m})',  (x2\_m, y2\_m),  textcoords="offset points",  xytext=(**0**, **10**),  ha='center',  color='blue', backgroundcolor="#eae1e196")      **def** **add\_hyperplane**(self, line\_color="#00A65D", mode="svc"):  """  points - результат работы метода get\_points\_of\_hyperplane  mode: 'svc', 'regression' or 'rbf'  """  X = np.concatenate((self.object\_1.points, self.object\_2.points), axis = **0**)  Y = np.array([**0**]\*len(self.object\_1.points) + [**1**]\*len(self.object\_2.points))  **if** mode == 'svc' **or** mode == 'regression':  **if** mode == 'svc':  C = **1.0** # параметр регуляризации SVM  clf = svm.SVC(kernel = 'linear', gamma=**0.7**, C=C)  **else**:  clf = LogisticRegression()    clf.fit(X, Y)    # Коэффициент признаков в решающей функции. (от тета 1 до тета n)  w = clf.coef\_[**0**]    # Перехват (также известный как смещение)  # добавлен в функцию принятия решения. (тета 0)  w0 = clf.intercept\_    # координаты линии по осям  xx = [np.min(X[:, **1**] - **5**), np.max(X[:, **1**] + **5**)]  yy = np.dot((-**1.**/w[**1**]), (np.dot(w[**0**],xx) + w0))    lM = mlines.Line2D(  xx, yy,  color=line\_color, linestyle="dotted"  )  self.ax.add\_line(lM)  **else**: # rbf  C = **1.0**  clf = svm.SVC(kernel = 'rbf', gamma=**0.7**, C=C )  clf.fit(X, Y)    h = .**02**  x\_min, x\_max = X[:, **0**].min() - **1**, X[:, **0**].max() + **1**  y\_min, y\_max = X[:, **1**].min() - **1**, X[:, **1**].max() + **1**  xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),  np.arange(y\_min, y\_max, h))    Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])    Z = Z.reshape(xx.shape)    self.ax.contour(xx, yy, Z, cmap=plt.cm.Paired) |

Листинг . Реализация класса distribution\_analysis

Наконец в Листинг 4 приведен код вызова «всего и вся».

|  |
| --- |
| **if** \_\_name\_\_ == "\_\_main\_\_":    # Создаем два "облака"  cloud\_1 = norm\_distribution(  N=**5000**,  math\_exp=[**1**, **100**],  cov\_m=[[**1**, **0**], [**0**, **200**]]  )    cloud\_2 = norm\_distribution(  N=**500**,  math\_exp=[**100**, **150**],  cov\_m=[[**10**, **0**], [**0**, **10**]]  )  # cloud\_1.show\_3d\_plot\_of\_distribution()  # Инициализация сравнения двух "облаков"  cloud\_comparison = distribution\_analysis(  cloud\_1, cloud\_2, graph\_title='Облака образов'  )  # Построение и отображение графика  # mode: 'svc', 'regression' or 'rbf'  cloud\_comparison.plot\_everything(show=True, hyperplane\_mode="regression") |

Листинг . Определение облаков и их сравнение

Как видно из Листинг 4, программа поддерживает несколько методов определения границ: метод опорных векторов с линейной и радиальной базисной функцией, а также метод линейной регрессии.

На Рис. 2, Рис. 3, Рис. 4 приведены результаты работы программы при различных параметрах и настройках:

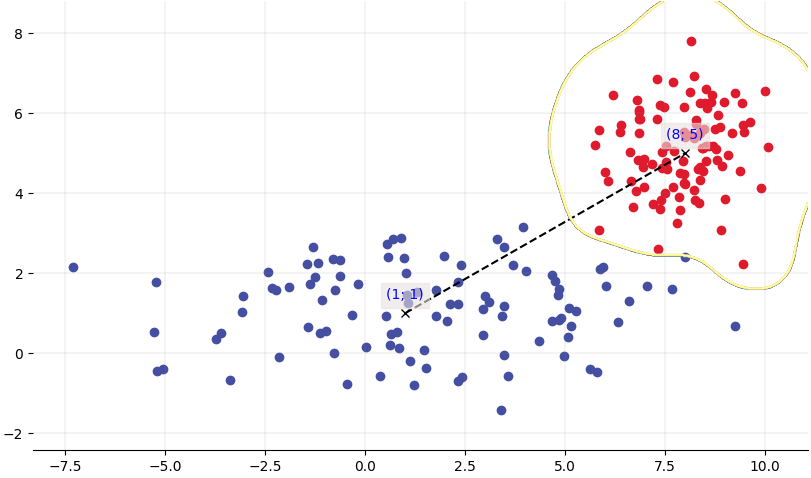


Рис. . Метод опорных векторов с радиальной БФ. N=100. Слева – cov\_m=[[10, 0], [0, 1]]; справа – cov\_m=[[1, 0], [0, 1]]

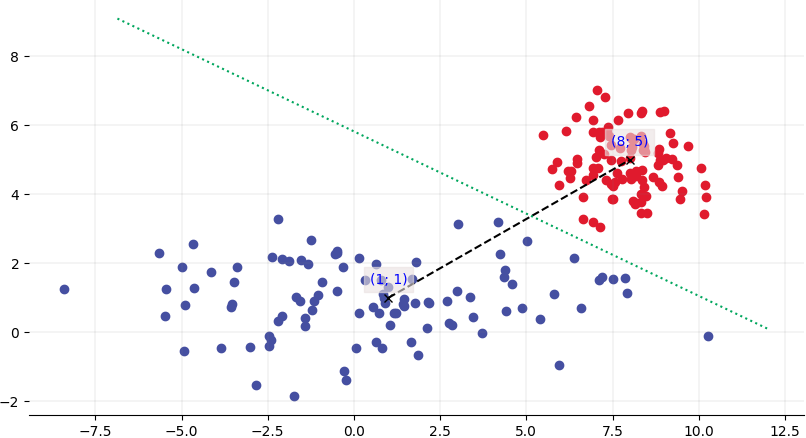


Рис. . Метод опорных векторов с линейной БФ. N=100. Слева – cov\_m=[[10, 0], [0, 1]]; справа – cov\_m=[[1, 0], [0, 1]]

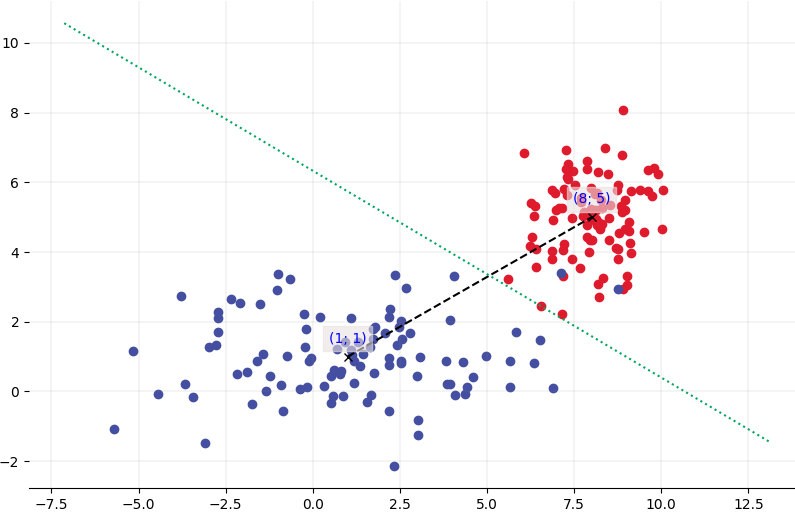


Рис. . Метод линейной регрессии. N=100. Слева – cov\_m=[[10, 0], [0, 1]]; справа – cov\_m=[[1, 0], [0, 1]]

# Вывод

В ходе лабораторной работы был изучена Р-модель распознавания и применены различные методы определения разделяющей гиперплоскости объектов 2 различных классов.