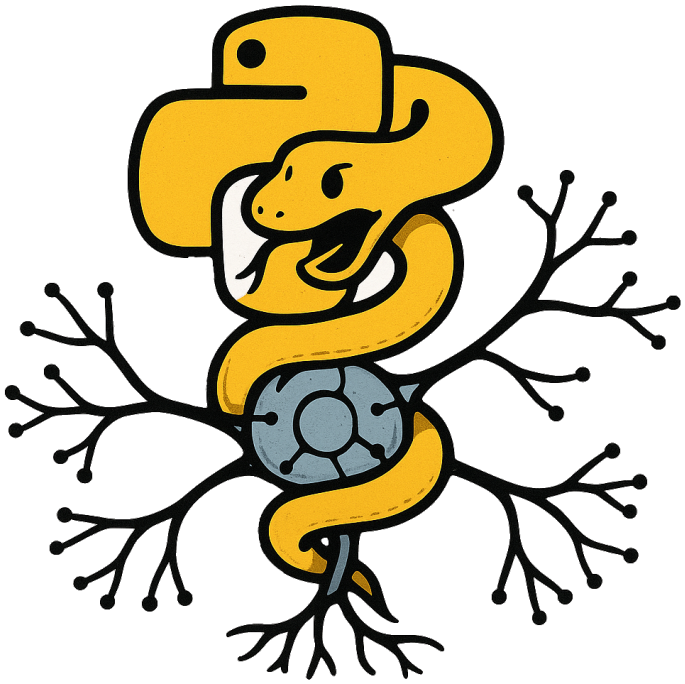
Основы Машинного Обучения



Автор: Марьичев Алексей

Нижегородский государственный университет им. Лобачевского

Предисловие

Словарь

- Вектор: $X = [x_1, x_2, ..., x_n]$
- Транспонированый вектор(обозначается ^{T}) : $X^T = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}$
-

Содержание:

Глава 1. Математический анализ

Глава 2. Линейная Алгебра

Глава 3. Теория Вероятности

Глава 4. Машинное обучение

Основные задачи ML:

- **Классификация** — определение объектов к определённым классам по общим признакам
- **Регрессия** — прогнозирование величин, функций или событий
- **Ранжирование** — упорядочивание входного набора данных

Раздел 1. Введение в Машинное обучение.

§1.1 Обучающая выборка

Представление объектов в виде различных векторов данных:

$$X = [x_1, x_2, \ldots, x_n]^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

X - вектор входных данных

Допустим, у нас дана матрица:

$$A = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{22} & \ldots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix}$$

Здесь A - матрица входных данных, n — количество признаков объекта, а m — количество самих объектов.
Таким же видом представлены и выходные данные:

$$Y = [y_1, y_2, \ldots, y_m]^T = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Y - вектор выходных данных

Теперь мы рассмотрим важный вопрос: как же такие объекты как изображения, звук и т. д. могут представляться в виде векторов?

Допустим, на вход задаче подается Изображение:

Рис. 1: Пример изображения

Теперь важное замечание — **размерность вектора n будет зависеть от количества пикселей в изображении**

Например, если изображение 1024 на 256, то размерность вектора будет $1024 \times 256 = 262144$

$$X = [x_1, x_2, \ldots, x_{262144}]$$

Теперь **объединим** эти понятия:

$X' = \{(x_i, y_i) \mid 0 < i \leq m\}$ — **размеченные данные (обучающая выборка)**

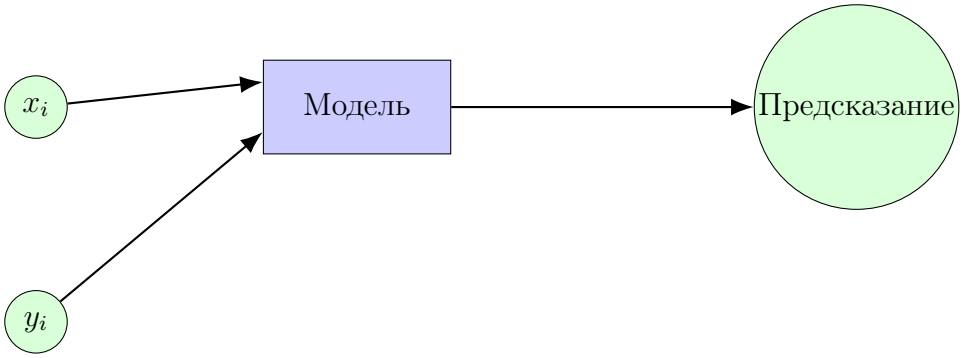
Это и является одним из важнейших понятий в области машинного обучения, с которым вы будете встречаться всюду.

---

§1.2 Постановка задачи для модели

А теперь разберемся с тем, как же модель будет "обучаться":

Допустим, у нас есть размеченные данные (x_i, y_i), которые подаются в некоторую модель



В результате из исходных данных мы получили некое предсказание, которое на первых этапах обучения может не иметь ничего общего с правильным ответом.

Теперь представим нашу модель как линейную функцию:

$$y(x) = \phi(x, \Delta) \quad (1)$$

Здесь Δ — **постоянно меняющийся параметр**
Его мы будем подстраивать для наиболее точного ответа нашей модели

Для лучшего понимания перейдем к задаче линейной регрессии. Задана функция:

$$y(x, k, b) = kx + b + \psi$$

Здесь k и b — параметры, от которых зависит угол поворота прямой, а так же ее сдвиг. Т.е. получается, что эта прямая может проходить как угодно, но за счет размеченных данных мы задаем модели желаемый результат:

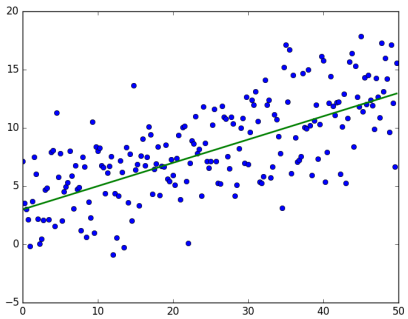


Рис. 2: Линейная регрессия

И получается, что во время обучения модель дает прогнозы все точнее и точнее к желаемому результату.

Но как же наш алгоритм понимает, что ответ надо корректировать?
Сейчас мы подошли к еще одному очень важному определению в области ML:

Функция потерь — функция, которая характеризует потери при неправильном предсказании модели
Примеры таких функций:

$$L(x, a) = |a(x) - y(x)| \quad \text{— абсолютная ошибка}$$

$$L(x, a) = (a(x) - y(x))^2 \quad \text{— квадратичная ошибка}$$

Также введем связное понятие:

Средний эмпирический риск

$$Q(a) = \frac{1}{l} \sum_{i=1}^{l} L(a(x_i), y_i)$$

Среднее значение функции потерь на обучающей выборке.

Это средне арифметическое по всем потерям в текущем цикле обучения модели.

Вспомним формулу (1): Наша задача — минимизировать средний эмпирический риск за счет изменения параметра Δ

§1.3 Линейная модель

Рассмотрим функцию $y = kx + b + \psi$.
В процессе обучения модели на данной функции перед нами будет стоять задача подобрать такие k' и b', чтобы сама функция y наиболее точно отображала желаемый результат.

Но что если мы попробуем выразить функцию через характеристики объекта?
Допустим, у нас есть ϕ_1, ϕ_2:

$$y = f_1(x)\phi_1 + f_2(x)\phi_2 + \psi$$

Здесь $f_1(x)$ — первая характеристика объекта, а $f_2(x)$ — вторая.
Очевидно, что если $f_1(x) = x$, $f_2(x) = 1$, а $\phi_1 = k'$, $\phi_2 = b'$, то формула сводится к изначальной:

$$y(x) = k'x + b'$$

Итак, линейная модель:

$$a(x) = \sum_{i=0}^{n} f_i(x)\phi_i$$

Рассмотрим конкретный пример

§1.3 BETA Переобучение

Нам дана функция:

$$f(x) = x^2 + x + 1$$

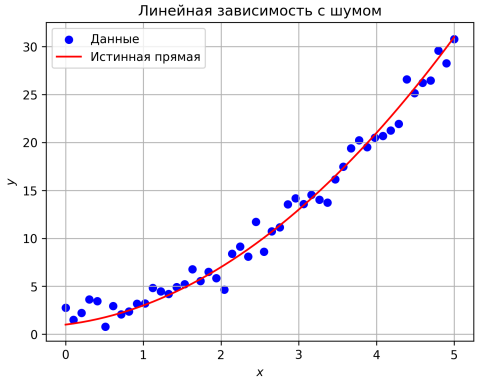


Рис. 3: Данные для функции

Сначала может показаться, что мы можем описать данную функцию с помощью одной характеристики: $a(x) = f(x)\phi$

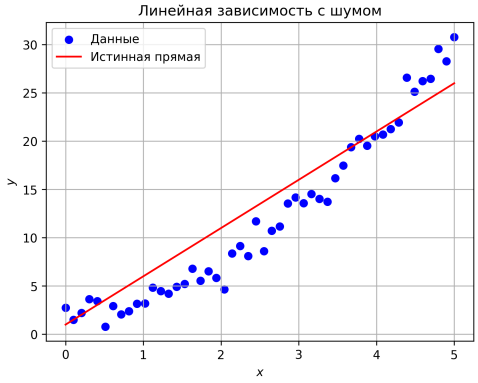


Рис. 4: Ошибка модели

Но в таком случае получится прямая линия, лишь по очертаниям похожая на нашу кривую.

В таком случае нам поможет полином:

$$\begin{cases} f_0(x) = \text{const} \\ f_1(x) = x \\ f_2(x) = x^2 \\ \vdots \\ f_n(x) = x^n \end{cases} \quad - \textbf{все характеристики}$$

Таким образом наша модель (при $f_0 = 1$):

$$a(x) = \sum_{i=0}^{n} f_i(x)\phi_i = \phi_0 + x\phi_1 + \ldots + x^n\phi_n$$

Важно

Система характеристик является **Линейно Независимой**

Почему характеристики не могут быть линейно зависимыми?

Допустим задана система:

$$\begin{cases} f_0(x) = 1 \\ f_1(x) = x \\ f_2(x) = x + 5 \end{cases}$$

Получается, что $f_2(x) = f_1(x) + 5$
Следовательно, если одну из характеристик можно выразить через другие, то зачем же она вообще нужна? Получается, что она просто является лишней в нашей системе и можно справиться без нее.

§1.4 Степень переобучения модели

1.4.1 Оценка по отложенной выборке (hold-out)

Для данного метода размеченные данные делят на две части:

- Обучающие
- Отложенная (hold-out)

Обычно данные делят в соотношении 70:30.
Цель: сравнить качество модели на данных, используемых при обучении, с новыми данными того же характера. Для этого строят два новых параметра: $Q(a, X)$ — **для обучающих данных и** $Q'(a, X')$ — **для отложенных данных.**

В результате, если средний эмпирический риск для обучающих данных меньше, чем для отложенных, то модель следует подкорректировать для лучшего показателя.

1.4.2 Скользящий контроль (leave-one-out)

Допустим, нам даны n различных размеченных данных: $X = (x_1, x_2, \ldots, x_n)$
Данный метод основан на том, что мы построим n таких моделей, что:

$$\begin{cases} a_1(x) : X_1 = (x_2, x_3, \ldots, x_n) \\ a_2(x) : X_2 = (x_1, x_3, x_4, \ldots, x_n) \\ \ldots \\ a_n(x) : X_n = (x_1, x_2, \ldots, x_{n-1}) \end{cases}$$

T.e. мы построили n различных моделей $a_i(x)$, таких, что каждая из них обучалась на наборе данных размерностью $n - 1$ (для a_i-ой модели убирали x_i-ый вектор данных)

Ну а конечная модель $a(x) = F(a_1, a_2, \ldots, a_n)$

На больших наборах данных этот способ требует огромной вычислительной мощи, потому он почти не используется на практике.

1.4.3 Кросс-валидация (cross-validation, k-fold)

Очень похожий на скользящий контроль метод, но различие состоит в том, что здесь мы разбиваем входные данные на некоторые группы и составляем из них модели:
Этот метод позволяет строить модели, которые будут обладать
лучшими обобщающими способностями, при меньшем количестве вычислений.

1.4.4 Обобщение моделей

В прошлых частях мы встречались с обобщением модели:
$a(x) = F(a_1(x), a_2(x), \ldots, a_n(x))$, но не упоминалось, как же это делается на самом

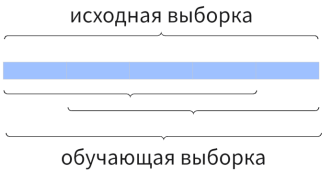


Рис. 5: Схема k-fold кросс-валидации

деле. В основном используют два метода для обобщения моделей, познакомимся с ними поближе:

1) **Выбор одной модели с лучшими показателями**

Допустим у нас есть модели:

$$\begin{cases} a_1(x) \\ a_2(x) \\ \dots \\ a_n(x) \end{cases}$$

Среди них мы выбираем ту модель, у которой
средний эмпирический риск минимальный. Но на самом деле, даже если этот показатель и наименьший, это не означает, что модель лучше остальных.

2) **Выбор наиболее часто встречающегося результата**

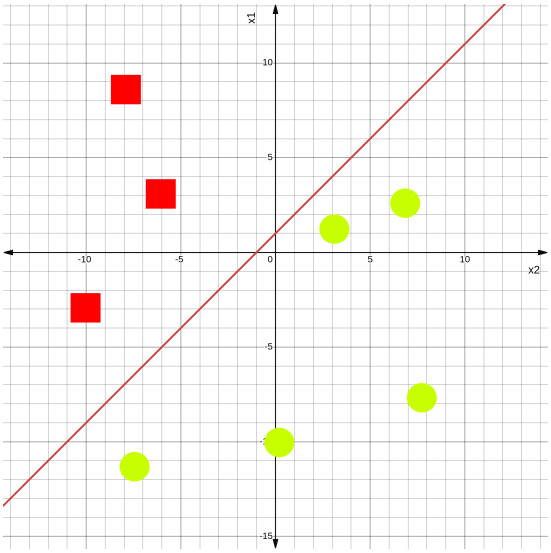
Допустим у нас есть модели, которые выдают определенный результат $\in (0, 1)$:

$$\begin{cases} a_1(x) = 1 \\ a_2(x) = 1 \\ \dots \\ a_{n-1}(x) = 1 \\ a_n(x) = 0 \end{cases}$$

Чаще всего встречается ответ 1, а значит мы его примем за верный.
Но данный способ **требует большой вычислительной мощи**, т.к. предсказания нам будет давать уже не одна, а n моделей.

§1.5 Уравнение гиперплоскости

Рассмотрим Здесь прямая в двумерном пространстве делит два класса предметов,

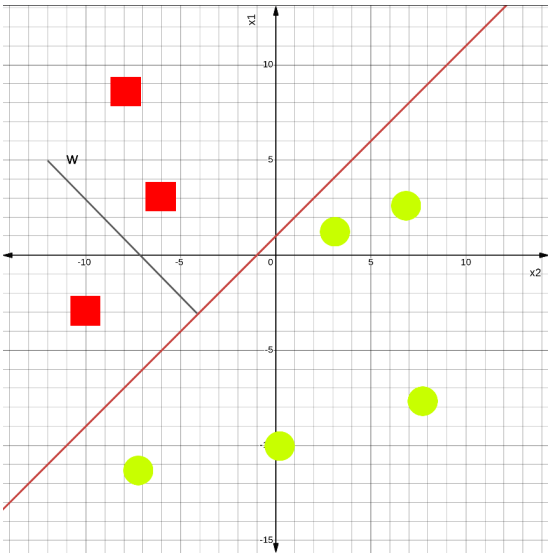


т.е. по левую часть от прямой располагаются предметы, относящиеся к одному классе, а по правую-к другому классу.

Обратимся к линейному уравнению:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

Вектор $w = (w_1, w_2)^T$ является нормалью к гиперплоскости, т.е. ортогонален всем векторам, лежащим в этой плоскости.



Докажем это:
Пусть x — произвольный вектор, лежащий в гиперплоскости. Тогда:

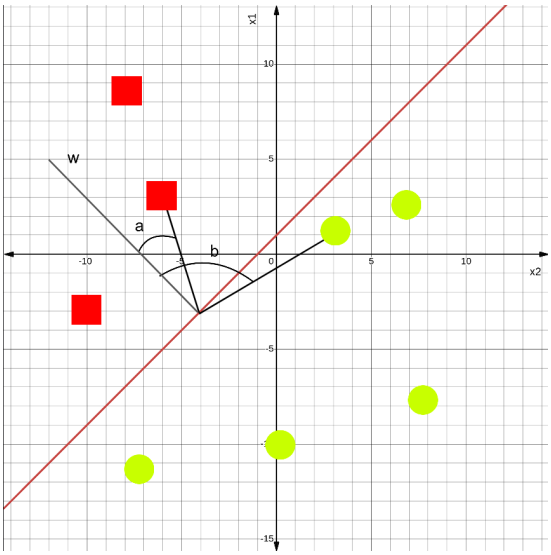
$$w \cdot x = \|w\|\|x\| \cos(\theta)$$

Поскольку x лежит в гиперплоскости, а w — нормаль, угол между ними 90°, следовательно:

$$\cos(\theta) = 0 \Rightarrow w \cdot x = 0$$

■

Но как же нам теперь отличать объекты одного класса от объектов другого класса с использованием полученных знаний?
Все очень просто, рассмотрим углы между ортогональной к плоскости прямой и прямой до объекта.



Заметим, что угол a является **острым углом**, как и любой другой угол между нормальную к гиперплоскости и прямой до объекта такого же класса.($\cos(a) > 0$)

а вот угол b является является **тупым**, как и любой другой угол между нормальную к гиперплоскости и прямой до объекта такого же класса.($\cos(a) < 0$)

А значит скалярное произведение (w, x) для одного класса будет положительным, а для другого отрицательным(следует из знака cos)
Более того, мы можем описать такую функцию как:

$$a(x, w) = sign((w, x)) = \begin{cases} -1, & (w, x) < 0 \\ 1, & (w, x) > 0 \end{cases}$$

При нуле объект не будет определен ни к одному классу.

§1.6 Задача бинарной классификации

Теперь перейдем к задаче на практике, нам заданы характеристики объектов:

№	Ширина	Длина	Жук
1	10	50	гусеница
2	20	30	божья коровка
3	25	30	божья коровка
4	20	60	гусеница
5	15	70	гусеница
6	40	40	божья коровка
7	30	45	божья коровка
8	20	45	гусеница
9	40	30	божья коровка
10	7	35	гусеница

Таблица 1: Характеристики насекомых по ширине и длине

Здесь обозначим -1 как гусеницу, 1 как божью коровку.

Критерий качества для разделяющей линии сформулировал Фрэнк Розенблатт, он определил это как количество неверных классификаций:

$$Q(a, X) = \sum_{i=1}^{n} [a(x_i) \neq y_i]$$

Квадратные скобки - индикатор ошибки, они переводят True/False в 1/0 соответственно(нотация Айверсона):

$$[a(x_i) \neq y_i] \in \{0, 1\}$$

В данной задаче мы можем посчитать критерий качества иначе, в виду того, что $y \in \{-1, 1\}$ Мы можем утверждать, что $y_i a(x_i))$ будет положительный при верной классификации и отрицательным при неверной.

$$Q(a, X) = \sum_{i=1}^{n} [y_i a(x)] < 0$$

Такое произведение очень часто используется в задачах бинарной классификации и обозначают как $M = y_i a(x_i)$ Его называют **Отступом**. Эта величина может показывать не только признак верной классификации, но и насколько далеко отстоит образ от разделяющей плоскости.

Теперь приступим к написанию такого алгоритма. У нас задано множество признаков, а так же правильная классификация, наша задача - это создать код, который от произвольного начального значения коэффициентов будет изменять эти значения в нужную сторону, пока не найдет тот, что верно подберет разделяющую прямую для нашей задачи(т.е. по критерию качества сумма будет равняться 0).

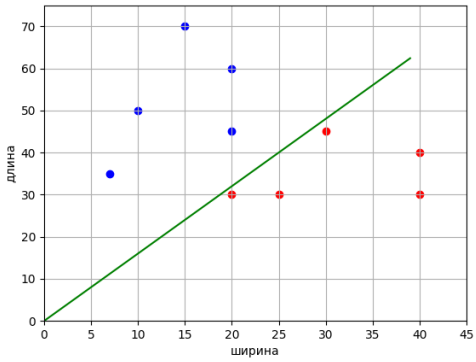
В данном случае у нас два признака, а потому вектор $W = [w_1, w_2]$ будет двумерным. Зафиксируем w_2 = -1 для простоты примера.

Реализация на Python:

```
import numpy as np
import matplotlib.pyplot as plt

x_train = np.array([[10, 50], [20, 30], [25, 30], [20, 60],
[15, 70], [40, 40], [30, 45], [20, 45], [40, 30], [7, 35]])
y_train = np.array([-1, 1, 1, -1, -1, 1, 1, -1, 1, -1])

n_train = len(x_train)
w = [0, -1]
a = lambda x: np.sign(x[0]*w[0] + x[1]*w[1])
N = 50
L = 0.1
e = 0.1                                          # небольшая добавка для w0

last_error_index = -1

for n in range(N):
    for i in range(n_train):                     # перебор по наблюдениям
        if y_train[i]*a(x_train[i]) < 0:
            w[0] = w[0] + L * y_train[i]
            last_error_index = i

    Q = sum([1 for i in range(n_train) if y_train[i]*a(x_train[i]) < 0])
    if Q == 0:       # показатель качества классификации (число ошибок)
        break        # останов, если все верно классифицируем

if last_error_index > -1:
    w[0] = w[0] + e * y_train[last_error_index]

print(w)

line_x = list(range(max(x_train[:, 0])))
line_y = [w[0]*x for x in line_x]

x_0 = x_train[y_train == 1]                      # формирование точек для 1-го
x_1 = x_train[y_train == -1]                     # и 2-го классов

plt.scatter(x_0[:, 0], x_0[:, 1], color='red')
plt.scatter(x_1[:, 0], x_1[:, 1], color='blue')
plt.plot(line_x, line_y, color='green')

plt.xlim([0, 45])
plt.ylim([0, 75])
plt.ylabel("длина")
plt.xlabel("ширина")
```

```
plt.grid(True)
plt.show()
```

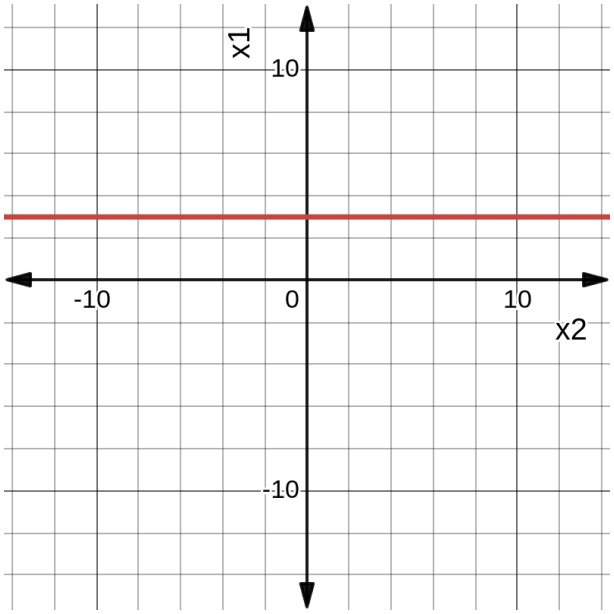
И после запуска получаем такой график:



Отлично! Мы провели классификацию двух объектов!

Решение задач

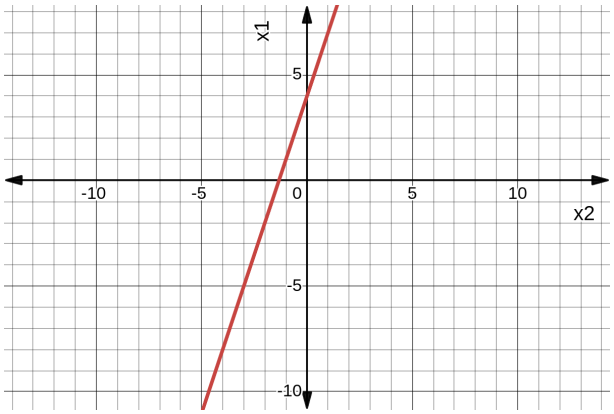
1) На графике представлена разделяющая линия в пространстве двух признаков



Требуется найти вектор коэффициентов $[w_0, w_1, w_2]^T$, удовлетворяющий линейной системе:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

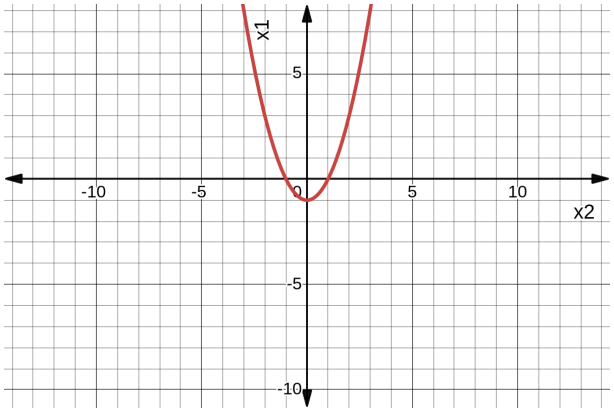
2) На графике представлена разделяющая линия в пространстве двух признаков



Требуется найти вектор коэффициентов $[w_0, w_1, w_2]^T$, удовлетворяющий линейной системе:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

3) На графике представлена разделяющая линия в пространстве двух признаков



Требуется найти вектор коэффициентов $[w_0, w_1, w_2]^T$, удовлетворяющий линейной системе:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$