Введение в линейную алгебру для машинного обучения

Цель занятия

После освоения темы:

- вы вспомните понятия вектора и матрицы, векторного пространства, нормы вектора, ортогональности и гиперплоскости;
- вспомните, как выполнять базовые операции над векторами и матрицами;
- сможете применять NumPy для работы с векторами и матрицами;
- сможете вычислять косинусную меру близости векторов и использовать ее для нахождения разницы между словами.

План занятия

1. Векторы. Основные операции над векторами
2. Матрицы. Основные операции над матрицами
3. Линейная алгебра в `NumPy`

Конспект занятия

1. Векторы. Основные операции над векторами

Для изучения машинного обучения необходимо качественное и глубокое понимание линейной алгебры. Именно поэтому изучение машинного обучения стоит начать именно с повторения основ.

Начнем с базовых понятий.

Скаляр — число в действительной числовой оси. Например, -2.

$\alpha \in R$ — скаляр

Вектор — упорядоченный набор чисел. Вектор может быть одномерный, двумерный, n-мерный. Здесь n — действительное число, не бесконечность. Бесконечномерные и комплекснозначные вектора рассматривать не будем.

$x \in R^n$ – вектор

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Примеры векторов:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \in \mathbb{R}^3, \quad \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \in \mathbb{R}^5$$

где R^3 — трехмерное пространство.

Матрица — набор векторов, поставленных либо вертикально (столбцы), либо горизонтально (строки).

$A \in R^{m \times n}$ — матрица

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Примеры матриц:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{2\times 2}, \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

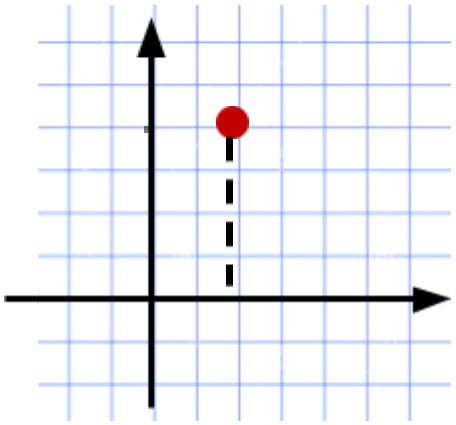
Эти понятия будем использовать практически на каждом занятии по машинному обучению. Векторы задают наши объекты (как правило, вектор-строки), матрицами будем задавать либо линейное преобразование, либо обучающую выборку. Скалярами — веса, параметры, коэффициенты.

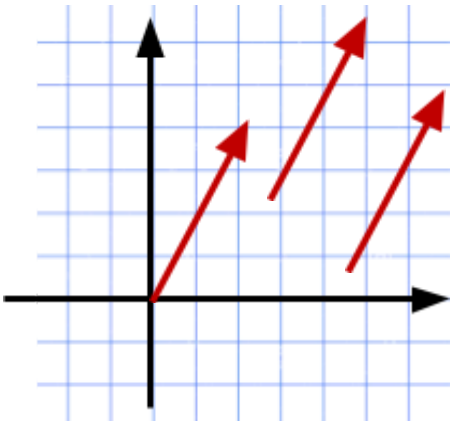
Физический смысл вектора. Вектор — набор упорядоченных чисел. Если мы рассмотрим декартовы координаты, то вектор — это точка в декартовых координатах. И в то же время вектору соответствует непосредственно вектор, направленный из начала координат в эту точку и перемещенный в любую другую точку.

Вектор задается направлением и длиной. Нам это понадобится в дальнейшем, потому что векторы мы будем рассматривать и как точки, и как направления.

Итак, вектор — это:

- упорядоченный набор чисел: $x = [1,\ 2]$;
- точка в декартовых координатах;
- направление + расстояние.





Далее мы будем работать с векторами в векторных, а чаще в линейных пространствах.

Векторное пространство — математическая структура, представляющая собой набор элементов, называемых векторами. Для них определены операции сложения друг с другом и умножения на число (скаляр).

Во-первых, для любой пары векторов мы можем получить их сумму, которая также будет принадлежать исходному множеству векторов:

$+\colon\ V \times V \to V$

Во-вторых, это операция умножения на скаляр. При умножении любого вектора на скаляр мы получаем другой вектор, но он также принадлежит исходному векторному пространству:

$\times: \ R \times V \to V$

Мы не можем выйти за пределы нашего множества векторов с помощью операций сложения или умножения. Эти операции удовлетворяют следующим свойствам:

	Свойство	**Пример**
1.	Ассоциативность +	$x + (y + z) = (x + y) + z$
2.	Коммутативность +	$x + y = y + x$
3.	Нейтральный элемент +	$\exists 0 \in V: \ \forall x \in V \quad 0 + x = x$
4.	Нейтральный элемент ·	$\forall x \in V \quad 1 \cdot x = x$
5.	Обратный элемент +	$\forall x \in V \ \exists - x \in V: \ x + (-x) = 0$
6.	Ассоциативность ·	$\alpha(\beta x) = (\alpha\beta)x$
7. 8.	Дистрибутивность	$(\alpha + \beta)x = \alpha x + \beta x$ $\alpha(x + y) = \alpha x + \alpha y$

Векторные операции

Сумма:

$$x = \left[x_1 \, x_2 \ \vdots \ x_n\right] \in R^n, \ y = \left[y_1 \, y_2 \ \vdots \ y_n\right] \in R^n$$

$$x + y = \left[x_1 + y_1 \, x_2 + y_2 \ \vdots \ x_n + y_2\right] \in R^n$$

Умножение на скаляр:

$$x = \left[x_1 \, x_2 \ \vdots \ x_n\right] \in R^n, \ \alpha \in R, \ \alpha x = \left[\alpha x_1 \, \alpha x_2 \ \vdots \ \alpha x_n\right] \in R^n$$

По сути, умножение на скаляр — это растяжение вектора вдоль его направления. Соотношение координат и наклон вектора при умножении на скаляр не меняется. Если умножить на отрицательное число, вектор будет развернут и будет смотреть в противоположную сторону, но все еще лежать на той же прямой.

Рассмотрим пример сложения и умножения в трехмерном пространстве:

$x, y \in R^3$:

$$x = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \qquad y = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$10x = \begin{bmatrix} 10 \\ 0 \\ 10 \end{bmatrix}$$

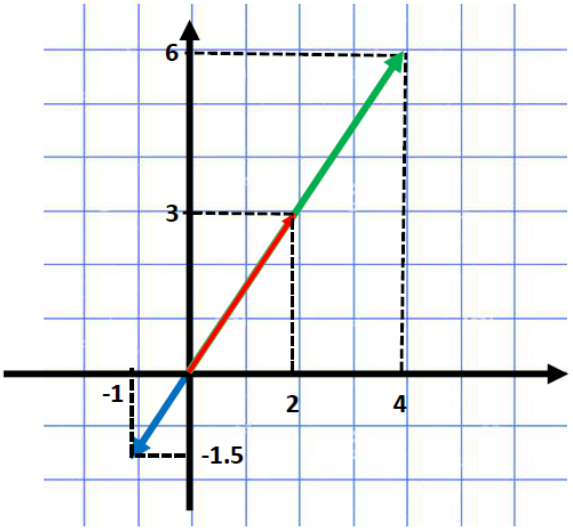
$$x + y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

Рассмотрим геометрические интерпретации векторных операций. Геометрическая интерпретация умножения на скаляр:

$\vec{a} = [2, 3]$

$2\vec{a} = [4, 6]$

$-0.5\vec{a} = [-1, -1.5]$

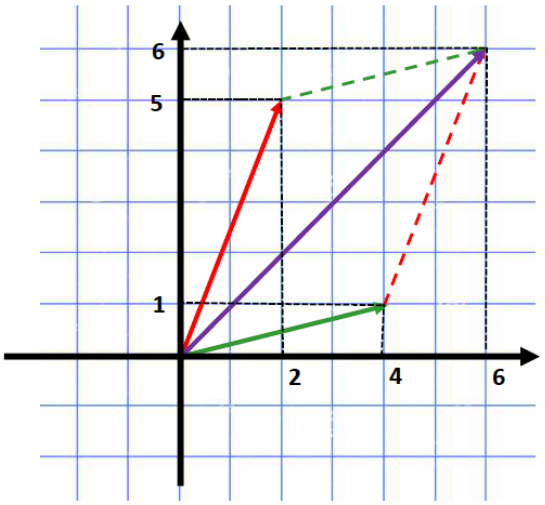


Геометрическая интерпретация сложения векторов:

$\vec{a} = [2, 5]$

$\vec{b} = [4, 1]$

$\vec{a} + \vec{b} = [6, 6]$



Вернемся к векторным пространствам.

Пример 1.

$\left(R^n, \ +, \ \cdot\right)$, $n \in N$ — векторное пространство, где

- сложение:
  $x + y = \left(x_1, x_2, \ldots, x_n\right) + \left(y_1, y_2, \ldots, y_n\right) = \left(x_1 + y_1, \ldots, x_n + y_n\right)$
- умножение на скаляр:
  $\alpha x = \alpha\left(x_1, x_2, \ldots, x_n\right) = \left(\alpha x_1, \alpha x_2, \ldots, \alpha x_n\right)$

Пример 2.

P^n – множество полиномов степени $\leq n$ с действ. коэф.

Например: $P^3 = \left\{ax^3 + bx^2 + cx + d \mid a, b, c, d \in R\right\}$

- Сумма:

$$\left(a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0\right) + \left(b_n x^n + b_{n-1}x^{n-1} + \ldots + b_0\right) =$$

$$= \left(a_n + b_n\right)x^n + \left(a_{n-1} + b_{n-1}\right)x^{n-1} + \ldots + \left(a_0 + b_0\right) \in P^n$$

- Умножение на скаляр:

$$\lambda\left(a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0\right) = \lambda a_n x^n + \lambda a_{n-1}x^{n-1} + \ldots + \lambda a_0 \in P^n$$

Аксиомы (1) – (8) выполнены → $\left(P^n, \ +, \ \cdot\right)$ **векторное пространство**

Скалярное произведение — это операция над двумя векторами, результатом которой является скаляр.

Свойства скалярного произведения:

- Симметричность: $\forall x, y \in V \ \langle x, y\rangle = \langle y, x\rangle$
- Неотрицательная определенность: $\forall x \in V \backslash \{0\} \ \langle x, x\rangle > 0$ и $\langle x, 0\rangle = 0$

Рассмотрим пример вычисления скалярного произведения:

$x = \left[x_1, \ldots, x_n\right], \ y = \left[y_1, \ldots, y_n\right] \in R^n$

$(x, y) = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n$

Например:

$x = [1,\ 2,\ 3,\ 4],\ \ y = [-\ 1,\ 0,\ 1,\ 2]$

$(x, y) = 1 \cdot (-\ 1) + 2 \cdot 0 + 3 \cdot 1 + 4 \cdot 2 = \ -\ 1 + 0 + 3 + 8 = 10$

Пространство $\left(R^n,\ +,\ \cdot\right)$ с указанным скалярным произведением $(\cdot, \cdot)$ называется **евклидовым**.

Нормой в векторном пространстве V называется функция $\|\ \cdot\ \|: V \to R$, сопоставляющая вектору $x \in V$ его длину $\|x\| \in R$.

Норма отображает вектор в число.

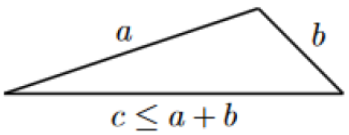
Свойства нормы:

1. $\forall x \in V,\ \ \forall \lambda \in R\ \ \|\lambda x\| = |\lambda| \|x\|$

При умножении вектора на скаляр мы можем вынести скаляр из самого вектора. Здесь используется модуль, потому что если мы поменяем направление вектора, длина остается той же.

2. Неотрицательно определена: $\forall x \in V\ \ \|x\| \geq 0$ and $\|x\| = 0 \Leftrightarrow x = 0$

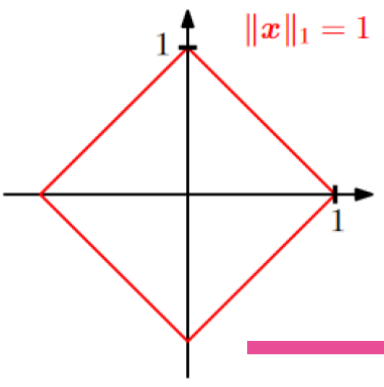
3. Неравенство треугольника: $\forall x, y \in V\ \ \|x + y\| \leq \|x\| + \|y\|$



Норму мы в основном будем использовать в двух целях — оценить норму вектора весов и оценить вектор ошибок. Во втором случае будем вычислять норму, чтобы посчитать, какую ошибку мы совершили при предсказании объектов.

Рассмотрим примеры норм.

Манхэттенская норма:



$x \in R^n$:

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

$\|[1,\ 2,\ 3]\|_1 = 1 + 2 + 3 = 5$

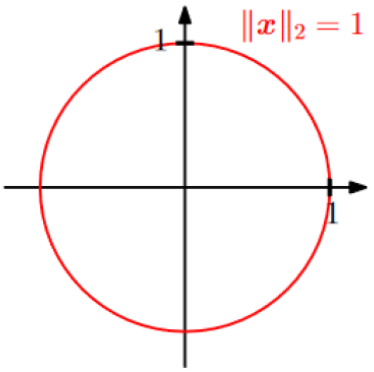
$\|[1,\ 0]\|_1 = 1 + 0 = 1$

$\|[-\ 1,\ 0]\|_1 = |-\ 1| + 0 = 1$

Ромбик на рисунке — на самом деле, сфера. Ведь сфера — это множество точек, равноудаленных от центра.

Давайте считать норму векторов (расстояние) как сумму модулей компонентов. Тогда получается, что все точки на ромбике равноудалены от центра, потому что сумма модулей их двух компонент порождает поверхность. Отсюда можно сделать вывод, что это сфера.

Евклидова норма:



$x \in R^n$:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

$\|[1,\ 2,\ 3]\|_2 = \sqrt{1^2 + 2^2 + 3^2} = \sqrt{14}$

$\|[1,\ 0]\|_2 = \sqrt{1^2 + 0} = 1$

$\|[-\ 1,\ 0]\|_2 = \sqrt{(-\ 1)^2 + 0} = 1$

Другие нормы:

Для $x = [x_1, \ldots, x_n] \in R^n$ l_p — норма задается как:

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}.$$

- l_1 — манхэттенская норма$\|\ \cdot\ \|_1$
- l_2 — евклидова норма $\|\ \cdot\ \|$
- l_∞: $\|\ x\ \|_\infty = |x_i|$

$\|[1,\ 2,\ 3]\|_\infty = 3, \quad \|[1,\ 0]\|_\infty = 1, \quad \|[-\ 1,\ 0.5]\|_\infty = 1$

Обратите внимание на l_∞, которая показывает максимальный элемент по модулю в нашем векторе. Все, кроме максимального элемента, мы игнорируем.

Свяжем скалярное произведение и норму. Во-первых, скалярное произведение может индуцировать (порождать) норму: $\|x\| := \sqrt{\langle x, x \rangle}$

Например, евклидова норма:

$$\sqrt{(x, x)} = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} = \|x\|_2$$

❗ **Не каждое скалярное произведение порождает норму.** Например, манхэттенская норма.

Рассмотрим **неравенство Коши-Шварца**. Скалярное произведение и индуцированную им норму связывает следующее неравенство:

$$|\langle x, y \rangle| \leq \|x\| \cdot \|y\|$$

Например, для евклидовой нормы:

$$|(x, y)| \leq \|x\|_2 \cdot \|y\|_2$$

Обратите внимание, что оно работает только если норма индуцирована скалярным произведением.

Расстояние между векторами. Расстояние между векторами, или метрика, задается как норма разницы векторов.

Расстояние между векторами x и y определяется как

$$d(x, y) := \|x - y\| = \sqrt{\langle x - y, x - y \rangle}$$

Для евклидовой нормы получается привычное евклидово расстояние

$$d(x, y) = \|x - y\|_2 = \sqrt{(x - y, x - y)} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

Углы между векторами

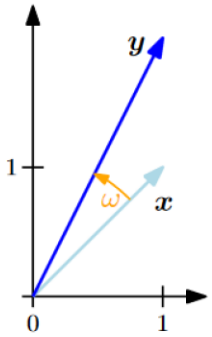
Угол между векторами может быть определен через скалярное произведение.

Согласно неравенству Коши-Шварца:

$|(x, y)| \leq \|x\| \cdot \|y\|,$

$-1 \leq \frac{(x,y)}{\|x\| \cdot \|y\|} \leq 1,$

ω: $cos\ \omega = \frac{(x,y)}{\|x\| \cdot \|y\|}$ — угол между x и y



Зная косинус между векторами, мы всегда можем посчитать угол между ними.

Вне зависимости от того, какова мера пространства, посчитать можно по введенной формуле.

Примеры расчета угла между векторами:

1. Найдем угол ω между $x = [5,\ 0]$ и $y = [1,\ 1]$:

$$\omega = arccos \frac{(x,y)}{\|x\|\|y\|} = arccos \frac{5 \cdot 1 + 0 \cdot 1}{\sqrt{5^2 + 0^2} \cdot \sqrt{1^2 + 1^2}} = arccos \frac{5}{5\sqrt{2}} = arccos \frac{\sqrt{2}}{4} = \frac{\pi}{4}$$

2. Найдем угол ω между $x = \left[\sqrt{3},\ 0, 1\right]$ и $y = [1,\ 0,\ 0]$:

$$\omega = arccos \frac{(x,y)}{\|x\|\|y\|} = arccos \frac{\sqrt{3}}{\sqrt{3+1} \cdot \sqrt{1}} = arccos \frac{\sqrt{3}}{2} = \frac{\pi}{6}$$

3. Найдем угол ω между $x = [1,\ 0,\ 0,\ 0,\ 1]$ и $y = [0,\ 1,\ 1,\ 0,\ 0]$:

$$\omega = arccos \frac{(x,y)}{\|x\|\|y\|} = arccos \frac{0}{\sqrt{2} \cdot \sqrt{2}} = arccos\ 0 = \frac{\pi}{2}$$

Ортогональность. Два вектора перпендикулярны или ортогональны друг другу, когда скалярное произведение между ними равно 0.

В зависимости от того, какое скалярное произведение мы выбираем, векторы будут по-разному ортогональны и не ортогональны друг другу.

Например, в $\mathbb{R}_n$ с заданным ранее скалярным произведением:

$x = [2,\ 3],\ \ y = [1,\ 0],\ \ (x, y) = 2 \neq 0 \ \rightarrow x$ и y не ортогональны.

$x = [1,\ 2,\ 3],\ \ y = [-\ 2, 1,\ 0],\ \ (x, y) = -\ 2 + 2 + 0 = 0 \ \rightarrow x$ и y ортогональны.

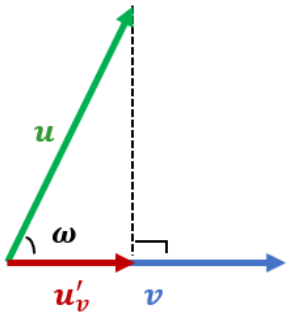
$x = [1,\ 0],\ \ y = [0, 1],\ \ (x, y) = 0,\ \ \|x\| = \|y\| = 1 \rightarrow x$ и y ортогональны.

Векторы ортогональны, только когда все компоненты при суммировании и умножении дают ноль.

Зачем ортогональность в машинном обучении. Понятие ортогональности в машинном обучении используется в методе главных компонент, который состоит в разложении пространства на ортогональные компоненты. Из них выбирается только некоторое подмножество, которое наиболее полезно для описания наших данных.

Угол между векторами в машинном обучении используется постоянно — например, косинус угла меры близости слов, эмбеддингов. При построении гиперплоскости нам также важно, на какой угол она отклоняется от чего-нибудь заранее заданного.

Ортогональная проекция. Пусть задана пара векторов u и v. $u_v^{'}$ — ортогональная проекция u на v.



Найдем длину $u_v^{'}$:

$0 \leq \omega \leq 90,$

$(u, v) = \|u\|\|v\| \cos cos\ \omega\ = \|u\|\|v\| \frac{\|u_v^{'}\|}{\|u\|} =$

$= \|u_v^{'}\|\|v\|$

Один вектор проецируем на другой, то есть берем только ту часть вектора *u*, которая сонаправлена вектору *v*. Причем неважно, в одну они сторону смотрят или в разную.

Рассмотрим случай вектора, который смотрит в другую сторону.

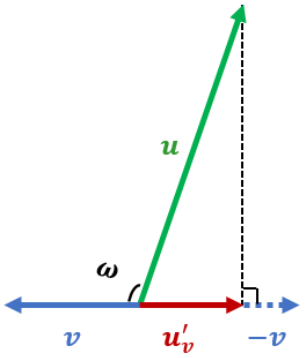
Пусть задана пара векторов u и v. $u_v^{'}$ — ортогональная проекция u на v.

Найдем длину $u_v^{'}$:

$90 \leq \omega \leq 180,$

$-\ (u, v) = \|u\|\|v\| cos\ \omega\ = \|u\|\|v\| \frac{\|u_v^{'}\|}{\|u\|} =$

$= \|u_v^{'}\|\|v\|$



В общем случае формула угла между векторами выглядит следующим образом.

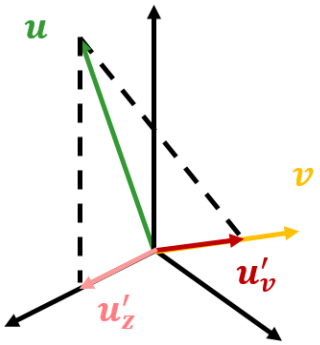
Пусть задана пара векторов u и v. $u_v^{'}$ — ортогональная проекция u на v.

Найдем длину $u_v^{'}$:

$|(u,v)| = \|u_v^{'}\|\|v\| \leftrightarrow \|u_v^{'}\| = \frac{|(u,v)|}{\|v\|}$,

$u_v^{'} = \frac{(u,v)}{(v,v)}v$

Рассмотрим примеры вычисления ортогональной проекции:



1. Найдем проекцию $u = [1,\ 3,\ 2]$ на $z = [0,\ 0,\ 1]$:

$u_z^{'} = \frac{(u,z)}{(z,z)}z = 2z = [0,\ 0,\ 2]$

2. Найдем проекцию $u = [1,\ 3,\ 2]$ на $v = [4,\ 1,\ 3]$:

$u_v^{'} = \frac{(u,v)}{(v,v)}v = \frac{4+3+6}{16+1+9}v = \frac{1}{2}v = [2,\ 0.5,\ 1.5]$

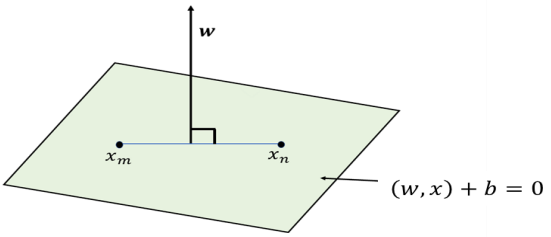
В машинном обучении мы будем использовать ортогональные проекции в задаче классификации, во всех линейных механизмах классификации, включая нейронные сети.

Гиперплоскость

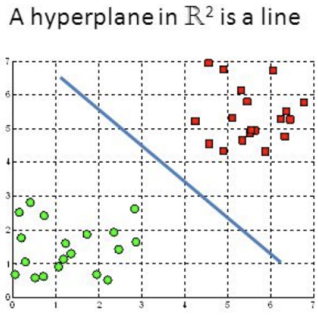
Гиперплоскость задается как $w_1x_1 + w_2x_2 + \ldots + w_nx_n + b = 0$, где хотя бы один элемент $w_i \neq 0$. В этой формуле x — все возможные координаты, а w — вектор нормали гиперплоскости.

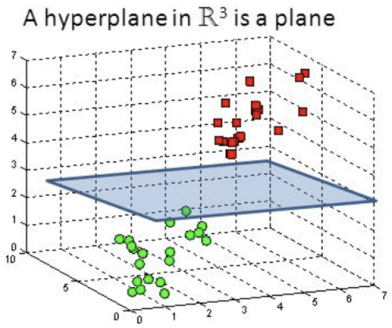
Более компактный вариант записи:

$(w,x) + b = 0,\ \ w = \left(w_1, \ldots, w_n\right)$.



Примеры гиперплоскостей:





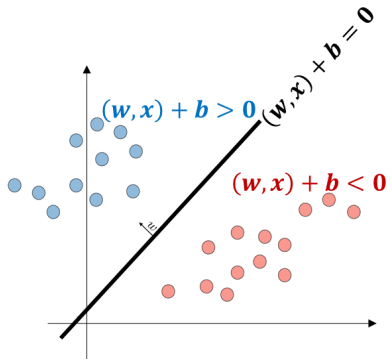
Здесь уже неявно видна задача классификации, но о ней мы поговорим позже.

Вектор $w = (w_1, ..., w_n)$ определяет **нормаль (нормальный вектор)**, он ортогонален каждому вектору на гиперплоскости.

Посмотрим на пример из машинного обучения. Будем называть каждую точку объектом. Объекты = 2D векторы. Мы хотим разделить их на классы +1 и −1. Для этого нам нужно построить гиперплоскость, которая разделяет их: сверху — +1, снизу — −1.

Более формально:

- Объекты над гиперплоскостью: $(w, x) + b > 0$.
- Объекты под гиперплоскостью: $(w, x) + b < 0$.



Считаем скалярное произведение $(w, x) + b$. Если оно больше 0, класс сверху синего цвета; если меньше 0, то класс снизу красного цвета.

2. Матрицы. Основные операции над матрицами

Матрица — множество векторов, которые упорядочены внутри одной матрицы.

$A \in R^{m \times n}$ – матрица с m строками и n столбцами:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{2\times 2}, \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

Будем рассматривать только действительнозначные матрицы.

Особые матрицы:

Диагональная	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$ $(a_{ii} \neq 0,\ a_{ij} = 0\ \forall i \neq j)$
Единичная	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $(a_{ii} = 1,\ a_{ij} = 0\ \forall i \neq j)$

Симметричная	$\begin{bmatrix}1 & 4 & 5\\ 4 & 2 & 6\\ 5 & 6 & 3\end{bmatrix}$ $(a_{ij} = a_{ji})$
Верхнетреугольная и нижнетреугольная	$\begin{bmatrix}1 & 2 & 3\\ 0 & 4 & 5\\ 0 & 0 & 6\end{bmatrix}, \begin{bmatrix}1 & 0 & 0\\ 2 & 3 & 0\\ 4 & 5 & 6\end{bmatrix}$ $(a_{ij} = 0\ \forall i > j\ or\ \forall i < j)$

Операции над матрицами

Введем операции сложения и умножения на скаляр:

- Сложение:

$$A = \{a_{ij}\}_{i=1,\dots,m,\ j=1,\dots,n},\ B = \{b_{ij}\}_{i=1,\dots,m,\ j=1,\dots,n},\ A + B = \{a_{ij} + b_{ij}\}_{i=1,\dots,m,\ j=1,\dots,n}$$

- Умножение на скаляр:

$$A = \{a_{ij}\}_{i=1,\dots,m,\ j=1,\dots,n},\ \lambda \in R,\ \lambda A = \{\lambda a_{ij}\}_{i=1,\dots,m,\ j=1,\dots,n}$$

Матрицы и операции сложения и умножения матриц на скаляр задают векторное пространство. Так что в каком-то смысле матрицу можно рассматривать как вектор.

Введем теперь **матричное умножение**. Матричное умножение постоянно используется в машинном обучении. Все нейронные сети, GPU (графические процессоры) необходимы, потому что умеют быстро считать умножение матриц.

Линейная модель — это умножение матрицы объект-признак на матрицу весов. Нейронная сеть — это умножение матрицы объект-признак на матрицу весов, только затем мы применяем активацию и повторяем многократно.

Итак, матричное умножение:

$$A = \{a_{ij}\}_{i=1,\dots,m,\ j=1,\dots,n},\ B = \{b_{ij}\}_{i=1,\dots,n,\ j=1,\dots,k}$$

$$A \cdot B = \{(A_i, B^j)\}_{i=1,\dots,m,\ j=1,\dots,k} = \left\{\sum_{l=1,\dots,n} a_{il} \cdot b_{lj}\right\}_{i=1,\dots,m,\ j=1,\dots,k}$$

❗ **Для умножения размерности матриц должны быть согласованы**

Например $R^{2\times2}$:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

❗ **Матричное умножение не коммутативно в общем случае: AB ≠ BA**

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 6 & 2 \\ 3 & 2 \end{bmatrix}, \quad AB = \begin{bmatrix} 30 & 14 \\ 12 & 6 \end{bmatrix}, \quad BA = \begin{bmatrix} 20 & 28 \\ 11 & 16 \end{bmatrix}$$

Умножение коммутативно в следующих случаях:

- Умножение на единичную матрицу E:

  $AE = EA = A$

- Умножение на матрицу из нулей O:

  $AO = OA = O$

Поговорим о **транспонировании матриц**. Операция транспонирования меняет местами строки и столбцы матрицы:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad A^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}.$$

Обратим внимание на следующие свойства:

- A – симметричная матрица $\Rightarrow A^T = A$
- $\left(A^T\right)^T = A$
- $(A + B)^T = A^T + B^T$
- $(AB)^T = B^T A^T$

Линейные преобразования

Введем линейное преобразование, которое воздействует на вектор:

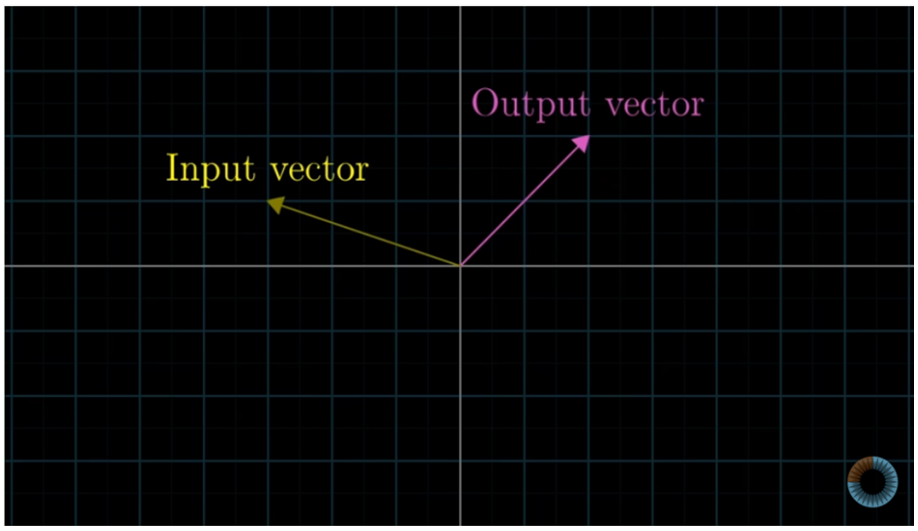
$x_{input} \rightarrow A \rightarrow x_{output}$

A – преобразование

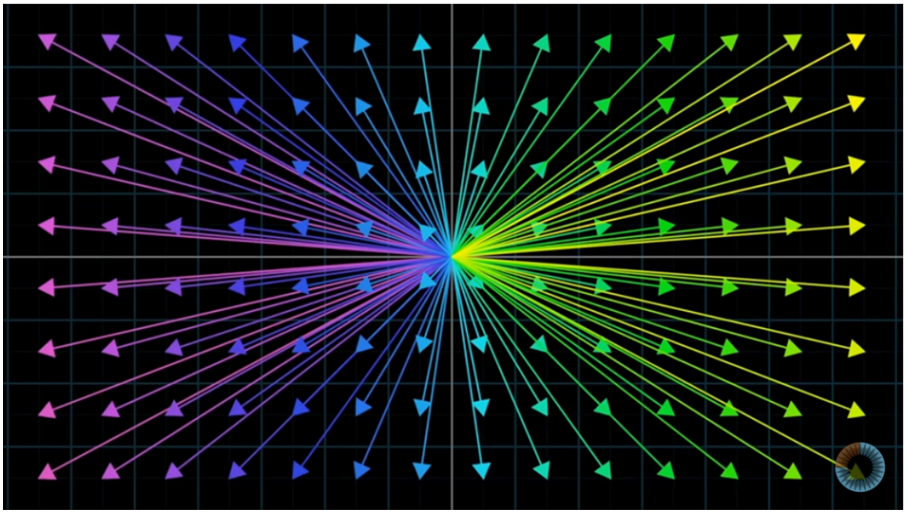
x_{input}, x_{output} – векторы

Пусть есть некий вектор. Мы воздействуем на него линейным преобразованием и получаем другой вектор в том же пространстве или в другом.

А что, если такое преобразование может называться матрицей? Как видите на рисунке, один вектор смотрит в одну сторону, другой в другую. Получается, мы его повернули, немного растянули, сжали.



Вы можете любой вектор, кроме нулевого, отобразить в любой из векторов на рисунке ниже и в любой между ними.



Свойства линейных преобразований:

1. $A(x + y) = A(x) + A(y)$
2. $A(\lambda x) = \lambda A x$

Линейное преобразование можно произвести и при помощи матриц:

$x_{input} = x_1 e_1 + x_2 e_2 + \ldots + x_n e_n$, $e_1, \ldots e_n$ – базис, $x_1, \ldots, x_n$ – координаты

$x_{output} = A(x_{input}) = A(x_1 e_1 + x_2 e_2 + \ldots + x_n e_n) = x_1 A(e_1) + x_2 A(e_2) + \ldots + x_n A(e_n)$

$A := [A(e_1) \mid A(e_2) \mid \ldots \mid A(e_n)]$

$\Rightarrow x_{output} = A(x_{input}) = A \cdot x_{input}$

Когда мы применяем линейное преобразование к вектору, мы по сути применяем его по отдельности к каждому из базисных векторов. Базисные векторы в ортогональном базисе ортогональны, поэтому они взаимодействуют независимо друг от друга. Итоговый результат — это преобразования над каждой из компонент по отдельности, совмещенные воедино. Таким образом получаем итоговый вектор.

Все линейные преобразования представляют собой поворот и растяжение-сжатие, а если меняем размерность — проекцию.

Рассмотрим пример линейного преобразования — поворот.

- Повернем векторы в R^2 на 90° против часовой стрелки.
- Базисные векторы повернутся:

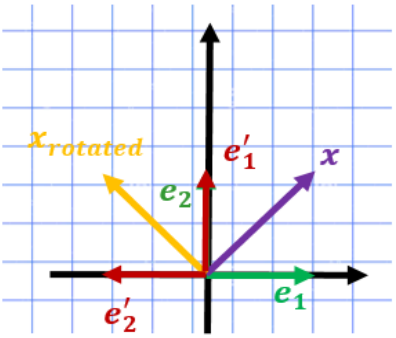
$$e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \to \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \to \begin{bmatrix} -1 \\ 0 \end{bmatrix}.$$

- Тогда матрица поворота:

$$R = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

- Повернем $x = [1,\ 1]^T$:

$$x_{rotated} = Rx = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$



Обратите внимание:

1. Каждое линейное преобразование задается матрицей.
   Столбцы = представление базисных векторов после преобразования.
2. И наоборот: каждая квадратная матрица задает некоторое линейное преобразование.

3. Линейная алгебра в `NumPy`

Поговорим, как можно справляться в Python с векторами и матрицами. В этом нам поможет библиотека `NumPy`.

Рассмотрим на примере векторного представления слов. Не будем погружаться в идею представления слов в виде векторов, это требует гораздо более глубокого погружения в мир машинного обучения. Воспользуемся готовыми результатами векторизации слов.

Открываем jupyter notebook:

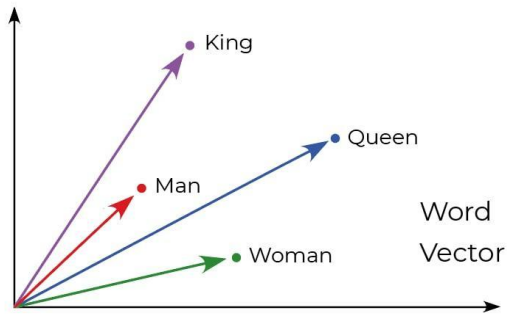


Рисунок иллюстрирует, что слова при отображении в векторное пространство могут не терять свои смысловые, семантические свойства. Пример двумерный. Видим, что слова King, Queen и Man, Woman похожи друг на друга. Угол между ними практически одинаковый. Это естественное свойство векторного представления слов.

Импортируем библиотеку:

```
import numpy as np
```

Зададим матрицу:

```
x = np.arange(16).reshape(4, 4)
```

Далее матрицу можно напечатать:

```
print("X :\n%s\n" % x)
```

Можно напечатать ее размерность:

```
print ("X.shape : %s\n" % (x.shape, ))
```

Можно применить поэлементное сложение:

```
print ("add x :\n%s\n" % (x + x))
```

Можно сложить матрицу с другой матрицей (в этом случае нужно подавать матрицы одинаковых размеров).

Можно использовать матричное умножение:

```
print ("X*X^T :\n%s\n" % np.natmul (x, x))
```

Можно посчитать среднее значение вдоль оси. «-1» — это то же самое, что последняя ось, «-2» — второй столбец с конца и т. д. Можно считать как с начала, так и с конца:

```
print ("mean over cols :\n%s\n" % (x.mean(axis=-1)))
```

Можно воспользоваться кумулятивной суммой:

```
print ("cumsum of cols :\n%s\n" % (np.cumsum(x,axis=0)))
```

Можно выполнить и другие операции из линейной алгебры (см. тьюториал Python Numpy Tutorial (with Jupyter and Colab)).

Если вы пользуетесь своей машиной, необходимо установить библиотеку `spacy`:

```
import spacy
```

Загружаем нужные векторы для английского языка:

```
nlp = spacy.load('en_core_web_lg')
```

Посмотрим на вектор слова dog. У него размерность 300:

```
len(nlp('dog').vector)
```

Можем измерить косинусную меры близости между векторами. Или же косинус угла. Вспомним, что это такое:

$sim(A, B) = cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$ – нормированное скалярное произведение между векторами.

```
def cosine(v1, v2):
       if np.linalg.norm(v1)*np.linalg.norm(v2) > 0:
           return np.dot(v1, v2)/(np.linalg.norm(v1)*np.linalg.norm(v2))
       else:
           return 0
```

Теперь можем посмотреть на разницу между словами dog и puppy (собака и щенок):

```
cosine(nlp('dog').vector, nlp('puppy').vector)
```

Также можно посмотреть угол между dog и kitten, между dog и coffee. Видим, что косинус угла постепенно уменьшается почти до $\frac{\pi}{2}$.

Все слова пишем с маленькой буквы.

Перейдем к чуть более сложной задаче. Обратимся к «Алисе в стране чудес». Это произведение будет источником слов на английском языке:

```
import requests
response = requests.get('https://www.gutenberg.org/files/11/11-0.txt')
```

Можем текст преобразовать в список токенов:

```
doc = nlp(response.text)
tokens = list(set([w.text for w in doc if w.is_alpha]))
```

Посмотрим, какие слова ближе всего друг к другу из нашего источника. Будем возвращать для заданного вектора список токенов, которые ближе к нему, по косинусной мере близости:

```
def spacy_closest(tokens, new_vec, n=10):
       return sorted(tokens,
                     key=lambda x: cosine(new_vec, nlp(x).vector),
                     reverse=True) [:n]
```

Посмотрим, какие слова близки к слову good. Обработка происходит долго, поскольку токенов очень много:

```
spacy_closest(tokens, nlp('good').vector
```

Можно посмотреть, как будут себя вести линейные операции над векторами. Посмотрим, что ближе всего к заданному вектору new_vec:

```
new_vec = nlp('king').vector - nlp('man').vector + nlp('woman').vector
spacy_closest(tokens, new_vec)
```

Скалярное произведение показывает, на сколько векторы похожи друг на друга.

Посмотрим на векторы различных предложений. Предложение — последовательность слов. Для каждого слова есть вектор. Мы можем их усреднить:

```
sent = nlp('My favorite food is strawberry ice cream.')
sentv = 0
for w in sent:
      sentv += w.vector
sentv /= len(sent)
sents = list(doc.sents)
```

Посмотрим, какие векторы ближе всего к заданному предложению:

```
def spacy_closest_sent(sentences, input_vec, n=10):
       return sorted(sentences,
                     key=lambda x: cosine(np.mean([w.vector for w in x], axis=0), input_vec),
                      reverse=True) [:n]

for s in spacy_closest_sent(sents, sentv, n=10):
      print (s)
      print ('\n- - -')
```

Строить векторное представление в предложении нужно чуть более умным образом, чем просто соединять все слова. С векторным представлением слов познакомимся, когда будем разбираться с рекуррентными нейронными сетями, трансформерами и мешком слов.

Numpy — это просто математический инструмент, мы будем постоянно им пользоваться. Создадим вектор из нулей и единиц:

```
a = np.ones(10)
b = np.zeros(10)
```

Вектора можно складывать:

```
a + b
```

Можно сложить со скаляром:

```
a + 23
```

Можно умножить:

```
a * b
```

Можно использовать скалярное произведение двух векторов:

```
a.dot(b)
```

Можно попытаться свернуть друг с другом, тогда нужно добавить операцию свертки.

`Numpy` — основной инструмент работы с матрицами. Потом дойдем до более сложных библиотек, которые могут делать с матрицами много полезных вещей.

Дополнительные материалы для самостоятельного изучения

1. Python Numpy Tutorial (with Jupyter and Colab) (cs231n.github.io)
2. Deep Learning (deeplearningbook.org)
3. Mathematics for Machine Learning (mml-book.github.io)