Модуль 2.1 Математические Основы для Data Science.

План:

1. Вспомним основные определения из линейной алгебры.
2. Попрактикуемся реализовывать вычисления в NumPy.
3. Q&A.

1. Линейная алгебра. Вектора и Матрицы.

Вектора

В линейной алгебре вектор - это элемент линейного пространства, который представляет с собой набор величин.

В геометрическом смысле, вектор - это направленный отрезок прямой в евклидовом пространстве.

$$\mathbf{x} = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 6 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} .3 \\ -7 \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} 1 & 4 & 5 & 6 \end{bmatrix}$$

2-е главных характеристики вектора:

- Длина (dimensionality, $\mathbb{R}^N$)
- Направление (orientation)

Вектор в Python

В Python вектор может быть представлен при помощи нескольких типов данных.

Самый простой способ - список. Но многие операции линейной алгебры не работают со списками.

Мы знакомы с библиотекой вычислений **NumPy**, которая является наилучшей при работе с объектами линейной алгебры.

Как создать вектор при помощи NumPy?

Вектор в Python

В Python вектор может быть представлен при помощи нескольких типов данных.

Самый простой способ - список. Но многие операции линейной алгебры не работают со списками.

Мы знакомы с библиотекой вычислений NumPy, которая является наилучшей при работе с объектами линейной алгебры.

Как создать вектор при помощи NumPy:

```
asList  = [1,2,3]
asArray = np.array([1,2,3]) # 1D array
rowVec  = np.array([ [1,2,3] ]) # row
colVec  = np.array([ [1],[2],[3] ]) # column
```

Операции над векторами

Сложение 2-х и более векторов.
Сложение определено только строго для векторов одинаковой размерности и одинаковой направленности.

$$\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} = \begin{bmatrix} 14 \\ 25 \\ 36 \end{bmatrix}$$

Операции над векторами

Сложение 2-х и более векторов.
Сложение определено только строго для векторов одинаковой размерности и одинаковой направленности.

$$\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} = \begin{bmatrix} 14 \\ 25 \\ 36 \end{bmatrix}$$

Вычитание 2-х и более векторов аналогично операции сложения векторов. Поэлементно проводим операцию вычитания.

$$\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} - \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} = \begin{bmatrix} -6 \\ -15 \\ -24 \end{bmatrix}$$

```
v = np.array([4,5,6])
w = np.array([10,20,30])
u = np.array([0,3,6,9])

vPlusW = v+w
uPlusW = u+w # error!
```

Операции над векторами в NumPy

Можем ли мы провести следующую операцию?

$$\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 10 & 20 & 30 \end{bmatrix} = ?$$

Операции над векторами в NumPy

Можем ли мы провести следующую операцию:

```
v = np.array([[4,5,6]]) # row vector
w = np.array([[10,20,30]]).T # column vector
v+w

>> array([[14, 15, 16],
          [24, 25, 26],
          [34, 35, 36]])
```

Как оказалось, да.

Результат такой операции в NumPy нарушает правила сложения двух векторов. Такая операция известна как **broadcasting** (меньший массив “дополняется” большим массивом, чтобы они имели совместимые формы).

https://numpy.org/doc/stable/user/basics.broadcasting.html

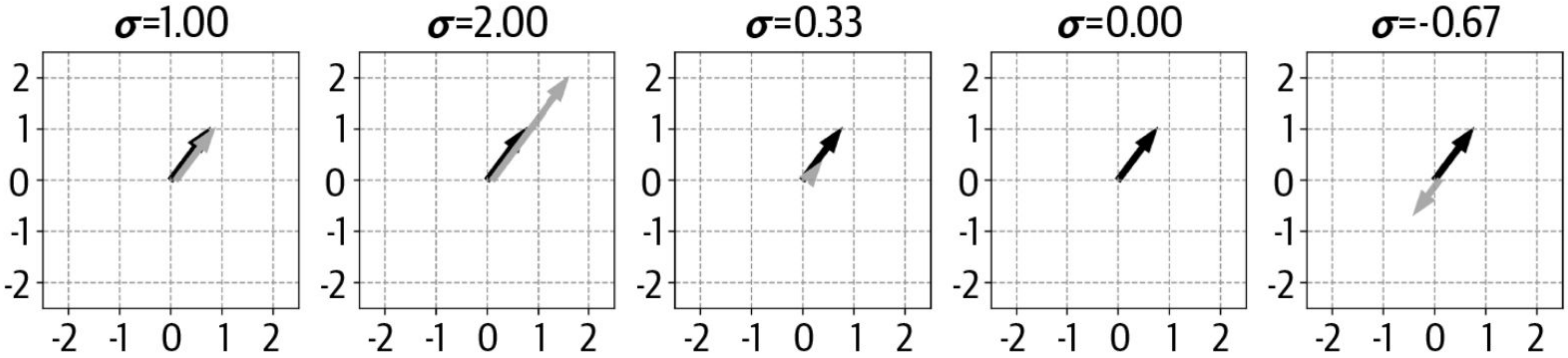
Операции над векторами

Умножение (сложение, вычитание) вектора на скаляр. Каждый элемент вектора умножается на скаляр.

$$\lambda = 4,\ \mathbf{w} = \begin{bmatrix} 9 \\ 4 \\ 1 \end{bmatrix}, \quad \lambda\mathbf{w} = \begin{bmatrix} 36 \\ 16 \\ 4 \end{bmatrix}$$

Результат умножения скаляра на список и на массив NumPy будут разными.

Геометрическая интерпретация скалярных действий с векторами



Операция транспонирования

Транспонирование преобразует векторы-столбцы в векторы-строки и наоборот. Данную операцию легко обобщить на транспонирование матриц.

Формально операция описывается как:

$$\mathbf{m}_{i,j}^{\mathrm{T}} = \mathbf{m}_{j,i}$$

Бродкастинг операций в NumPy

Бродкастнг (Broadcasting) - операция существующая только в компьютерной линейной алгебре и компьютерных вычислениях.

Бродкастинг означает многократное повторение операции между одним вектором и каждым элементом другого вектора, например:

```
v = np.array([[1,2,3]]).T # col vector
w = np.array([[10,20]])   # row vector
v + w # broadcasting

>> array([[11, 21],
          [12, 22],
          [13, 23]])
```

!Бродкастинг позволяет проводить эффективные и компактные вычисления в компьютерных вычислениях. Примеры бродкастинга мы увидим в алгоритмах кластеризации

Бродкастинг операций в NumPy

Бродкастнг (Broadcasting) - операция существующая только в компьютерной линейной алгебре и компьютерных вычислениях.

Бродкастинг означает многократное повторение операции между одним вектором и каждым элементом другого вектора, например:

```
v = np.array([[1,2,3]]).T # col vector
w = np.array([[10,20]])   # row vector
v + w # broadcasting

>> array([[11, 21],
          [12, 22],
          [13, 23]])
```

Длина вектора и единичный вектор

Длина вектора (magnitude), также известная как **норма** вектора - длина направленного отрезка, определяющего вектор, то есть расстояние между началом и концом вектора, рассчитанная при помощи стандартной Евклидовой формулы расстояния:

Длина вектора и единичный вектор

Длина вектора (magnitude), также известная как **норма** вектора - длина направленного отрезка, определяющего вектор, то есть расстояние между началом и концом вектора, рассчитанная при помощи стандартной Евклидовой формулы расстояния:

$$\| \mathbf{v} \| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Расчет длины вектора в NumPy:

```
v = np.array([1,2,3,7,8,9])
v_mag = np.linalg.norm(v)
```

Длина вектора и единичный вектор

В некоторых расчетах, нам будет необходим вектор, геометрическая длина которого равна единице.

Такой вектор называется единичным вектором и обозначается как $\| \mathbf{v} \| = 1$.

Примеры расчетов, в которых нам понадобится единичный вектор: ортогональные матрицы, матрицы вращения и поворота, собственные векторы и сингулярные векторы.

Как найти единичный вектор любого вектора в линейном пространстве?

Длина вектора и единичный вектор

В некоторых расчетах, нам будет необходим вектор, геометрическая длина которого равна единице.

Такой вектор называется единичным вектором и обозначается как $\| \mathbf{v} \| = 1$.

Примеры расчетов, в которых нам понадобится единичный вектор: ортогональные матрицы, матрицы вращения и поворота, собственные векторы и сингулярные векторы.

Как найти единичный вектор любого вектора в линейном пространстве:

$$\widehat{\mathbf{v}} = \frac{1}{\| \mathbf{v} \|}\mathbf{v}$$

Скалярное произведение векторов

Скалярное произведение векторов (dot product, inner product) одна из важнейших операций в линейной алгебре. Является строительным блоком многих операций и алгоритмов: сверточная операция, корреляция, преобразование Фурье, умножение матриц, извлечение линейных признаков, фильтрация сигналов и т.д.

$$\mathbf{a}^{\mathrm{T}}\mathbf{b} \quad \mathbf{a}\cdot\mathbf{b} \quad \langle\mathbf{a},\mathbf{b}\rangle$$

Результат скалярного произведение - скаляр, который предоставляет информацию об отношении двух линейных объектов.

Как вычисляется скалярное произведение векторов?

Скалярное произведение векторов

Скалярное произведение векторов (dot product, inner product) одна из важнейших операций в линейной алгебре. Является строительным блоком многих операций и алгоритмов: сверточная операция, корреляция, преобразование Фурье, умножение матриц, извлечение линейных признаков, фильтрация сигналов и т.д.

Результат скалярного произведение - скаляр, который предоставляет информацию об отношении двух линейных объектов.

$$\mathbf{a}^{\mathrm{T}}\mathbf{b} \quad \mathbf{a} \cdot \mathbf{b} \quad \langle \mathbf{a}, \mathbf{b} \rangle$$

Как вычисляется скалярное произведение векторов: $\delta = \sum_{i=1}^{n} \mathrm{a}_i \mathrm{b}_i$

Поэлементное умножение элементов векторов и их сумма. В модуле NumPy скалярное произведение реализовано при помощи метода **np.dot(v1, v2)**:

```
v = np.array([1,2,3,4])
w = np.array([5,6,7,8])
np.dot(v,w)
```

Скалярное произведение векторов. Значение

Хорошо, вспомнили что такое скалярное произведение, но что это означает, какую информацию несет с собой скалярное произведение?

$$\mathbf{a}^{\mathrm{T}}\mathbf{b} \quad \mathbf{a} \cdot \mathbf{b} \quad \langle \mathbf{a}, \mathbf{b} \rangle$$

Скалярное произведение векторов. Значение

Хорошо, вспомнили что такое скалярное произведение, но что это означает, какую информацию несет с собой скалярное произведение?

Скалярное произведение можно интерпретировать как меру сходства между линейными объектами.

Величина скалярного произведения зависит от масштаба данных, что означает, что скалярное произведение между данными, измеряемыми в граммах и сантиметрах, будет больше чем скалярное произведение между данными измеренными в тоннах и метрах.

$$\mathbf{a}^{\mathrm{T}}\mathbf{b} \quad \mathbf{a}\cdot\mathbf{b} \quad \langle\mathbf{a},\mathbf{b}\rangle$$

Эту погрешность на масштабе можно устранить за счет коэффициента нормализации. В действительности **нормализованное скалярное** произведение между двумя линейными объектами называется **коэффициентом корреляции Пирсона**, который мы широко будем использовать в статистическом анализе.

Скалярное произведение векторов. Свойства

- Скалярное произведение дистрибутивно

$$\mathbf{a}^{\mathrm{T}}(\mathbf{b}+\mathbf{c}) = \mathbf{a}^{\mathrm{T}}\mathbf{b} + \mathbf{a}^{\mathrm{T}}\mathbf{c}$$

- Скалярное произведение коммутативно

$$\overline{a} \cdot \overline{b} = \overline{b} \cdot \overline{a}$$

- Скалярное произведение ассоциативно

$$(\lambda\overline{a}) \cdot \overline{b} = \lambda(\overline{b} \cdot \overline{a})$$

- Если скалярное произведение двух ненулевых векторов равно нулю, то эти векторы ортогональны (перпенидкулярны)

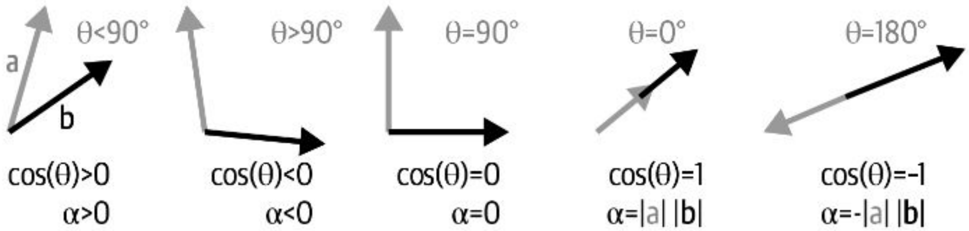
$$a \neq 0, b \neq 0, a \cdot b = 0 <=> a^{\perp} b$$

Геометрия скалярного произведения

Существует также геометрическое определение скалярного произведения, которое представляет с собой произведение величин двух векторов, масштабированное на косинус угла между ними:

$$\alpha = \cos\left(\theta_{\mathbf{v},\mathbf{w}}\right) \| \mathbf{v} \| \| \mathbf{w} \|$$

Соответственно интерпретация геометрического скалярного произведения:



Поэлементное умножение двух векторов

Поэлементное умножение двух векторов (Hadamard multiplication, произведение Адамара) - бинарная операция над двумя векторами (матрицами) одинаковой размерности, результатом которой является матрица той же размерности, где каждый элемент - это поэлементное произведение исходных векторов.

$$\begin{bmatrix}5\\4\\8\\2\end{bmatrix} \odot \begin{bmatrix}1\\0\\.5\\-1\end{bmatrix} = \begin{bmatrix}5\\0\\4\\-2\end{bmatrix}$$

Реализована для удобства вычислений.

Векторное произведение

Векторное произвдение (outer product) - метод умножения вектора-столбца и вектора-строки.

Каждый столбец в матрице произведения представляет собой скаляр вектора-столбца, умноженный на соответствующий элемент вектора-строки.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \begin{bmatrix} d & e \end{bmatrix} = \begin{bmatrix} ad & ae \\ bd & be \\ cd & ce \end{bmatrix}$$

Реализация векторного умножения в NumPy:

- np.outer()
- np.dot()

Ортогональная декомпозиция векторов

Декомпозиция вектора (матрицы) означает “разбиение” вектора (матрицы) но несколько более простых частей. Декомпозиция (разложение) используется для выявления информации, которая “скрыта” в векторе (матрице) для более эффективной и оптимальной работы с линейными объектами.

Ортогональная декомпозиция (разложение) - разбиение вектора на два отдельных вектора, один из которых ортогонален опорному вектору, а другой параллелен ему.

Ортогональное векторное разложение напрямую приводит к процедуре Грама-Шмидта или QR-разложению, которое очень часто используется для решений задач статистического анализа.

https://www.math.ucla.edu/~yanovsky/Teaching/Math151B/handouts/GramSchmidt.pdf

Ортогональная декомпозиция векторов

Существует 2-а вектора **a** и **b** расположенных в начальных координатах евклидова пространства.

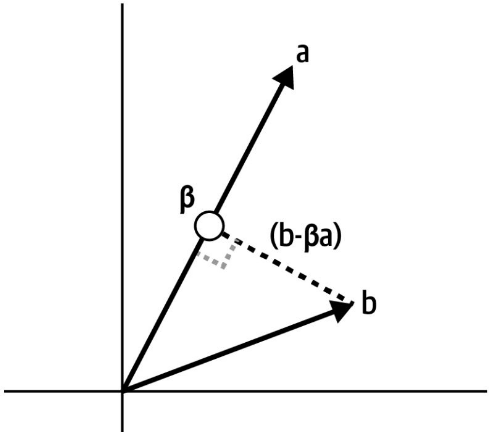
Наша цель - найти точку на векторе **a** которая как можно ближе находится к концу вектора **b**.

Данный подход выражается через задачу оптимизации: спроецировать вектор **a** на вектор **b** таким образом, чтобы **расстояние проекции было минимальным**. Такая точка будет являться масштабированной версией вектора **a** то есть **βa**. Теперь необходимо найти скаляр **β**.

Мы можем воспользоваться операцией (**b - βa**) чтобы найти данный вектор, пусть будет **c**.

Ключевым моментом здесь является то, что точка на векторе **a** которая расположена ближе всего к концу вектора **b** находится путем проекции вектора под оптимальным углом.

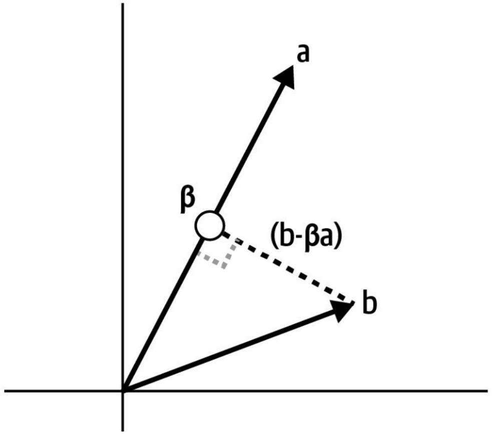
Интуитивно, представьте треугольник, длина отрезка **b** до **βa** больше если угол ∠**βa** становится меньше 90°



Ортогональная декомпозиция векторов

Сложив всю логику вместе, нам необходимо решить:

$$\mathbf{a}^{\mathrm{T}}\left(\mathbf{b}-\beta\mathbf{a}\right)=0$$



Используя знания о векторных операциях выразим **β**:

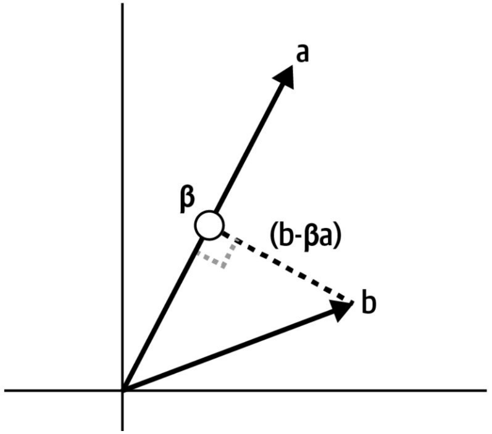
Ортогональная декомпозиция векторов

Сложив всю логику вместе, нам необходимо решить:

$$\mathbf{a}^{\mathrm{T}}(\mathbf{b} - \beta\mathbf{a}) = 0$$

Используя знания о векторных операциях выразим **β**:

$$\begin{aligned} \mathbf{a}^{\mathrm{T}}\mathbf{b} - \beta\mathbf{a}^{\mathrm{T}}\mathbf{a} &= 0 \\ \beta\mathbf{a}^{\mathrm{T}}\mathbf{a} &= \mathbf{a}^{\mathrm{T}}\mathbf{b} \\ \beta &= \frac{\mathbf{a}^{\mathrm{T}}\mathbf{b}}{\mathbf{a}^{\mathrm{T}}\mathbf{a}} \end{aligned}$$



Таким образом мы нашли формулу проекции точки на прямую с минимальным расстоянием.

Это называется ортогональной проекцией и является базой для многих вычислений в статистическом анализе и машинном обучении, включая в будущем метод наименьших квадратов для решения линейных регрессионных моделей.

Векторное пространство

Коллекция векторов в линейном пространстве называется **векторным множеством (vector set либо просто векторным пространством)** и обозначается следующим образом:

$$V = \{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$$

Зачем нам векторное множество:

представьте, что у вас датасет с миллионом наблюдений позитивных и негативных случаев заболеванием Covid-19, вы можете хранить данные для каждой страны в трехмерном векторе: [страна, результат, исход] либо создать векторное множество содержащее миллион этих записей.

Векторные множества, как вы уже поняли, могут содержать конечное и бесконечное количество векторов.

Кроме того, векторное множество может быть пустым $\boldsymbol{V = \{\ \}}$

Линейная комбинация векторов

Линейная комбинация (**linear weighted combination**) - это способ, который позволяет учесть информацию из нескольких признаков, при этом некоторые признаки вносят больший вклад, чем другие.

Такую фундаментальную операцию также называют **linear mixture** или взвешенной линейной комбинацией.

Линейная комбинация векторов - это операция скалярного умножения векторов и их сумма, результатом которой является вектор: $\mathbf{w} = \lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2 + \ldots + \lambda_n \mathbf{v}_n$

Пример результата линейной комбинации векторов:

$$\lambda_1 = 1,\ \lambda_2 = 2,\ \lambda_3 = -3, \quad \mathbf{v_1} = \begin{bmatrix} 4 \\ 5 \\ 1 \end{bmatrix},\ \mathbf{v_2} = \begin{bmatrix} -4 \\ 0 \\ -4 \end{bmatrix},\ \mathbf{v_3} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

$$\mathbf{w} = \lambda_1 \mathbf{v_1} + \lambda_2 \mathbf{v_2} + \lambda_3 \mathbf{v_3} = \begin{bmatrix} -7 \\ -4 \\ -13 \end{bmatrix}$$

Линейная комбинация векторов

Зачем нужна:

- Прогнозируемые данные из модели машинного обучения выводятся путем взятия линейной комбинации регрессоров (переменных предикторов) и коэффициентов (скаляров), которые вычисляются с помощью алгоритма наименьших квадратов (МНК) в задачах регрессии.

- В задачах по уменьшению размерности, например PCA, каждый компонент выводится как взвешенная линейная комбинация признаков данных с весами (коэффициентами), оптимизированными на максимизацию дисперсии компонента.

- Модели нейронных сетей включают 2 основные операции - линейную комбинацию входных данных с последующим их нелинейным преобразованием. Далее веса подбираются путем минимизации функции ошибки, которая обычно представляет с собой разницу между предсказанием модели и реальной целевой переменной.

Линейная независимость

Множество векторов **линейно зависимо**, если хотя бы один вектор во множестве **может быть выражен как линейная комбинация других векторов** в данной группе.

Верно и обратное, множество векторов **линейно незивисимо, если ни один вектор не может быть представлен как линейная комбинация** других векторов в данном множестве.

Попробуйте определить зависимость (независимость) множества векторов:

$$V = \left\{ \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \end{bmatrix} \right\} \qquad S = \left\{ \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 6 \end{bmatrix} \right\}$$

$$T = \left\{ \begin{bmatrix} 8 \\ -4 \\ 14 \\ 6 \end{bmatrix}, \begin{bmatrix} 4 \\ 6 \\ 0 \\ 3 \end{bmatrix}, \begin{bmatrix} 14 \\ 2 \\ 4 \\ 7 \end{bmatrix}, \begin{bmatrix} 13 \\ 2 \\ 9 \\ 8 \end{bmatrix} \right\}$$

Линейная независимость

Как определить линейную независимость на практике.

Алгоритм определения довольно прост. Кто помнит?

Линейная независимость

Как определить линейную независимость на практике.

Алгоритм определения довольно прост:

Способ определения линейной независимости состоит в том, чтобы:

1. Создать матрицу из набора векторов
2. Вычислить ранг матрицы
3. Сравнить ранг матрицы с меньшим из числа строк или столбцов

Подробнее про ранг матрицы поговорим дальше.

Запомните пока, что **независимость - это свойство набора векторов**, а не одного конкретного вектора в наборе.

Линейная независимость

Теперь более формально, линейная зависимость выражается как:

$$\mathbf{0} = \lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2 + \ldots + \lambda_n \mathbf{v}_n, \quad \lambda \in \mathbb{R}$$

То есть, линейная зависимость означает, что мы можем определить некоторую взвешенную линейную комбинацию множества векторов, комбинация которых дает нам нулевой вектор.

Если мы можем найти такие **α*v***, которые будут удовлетворять условию уравнения, то наше множество векторов будет линейно зависимо (верное и обратное).

Базис

Базисом плоскости называется **пара линейно независимых** (неколлинеарных) векторов **(e_1, e_2)**, взятых в определенном порядке, при этом **любой вектор плоскости является линейной комбинацией базисных векторов**.

Важным моментом определения является тот факт, что векторы взяты в **определенном порядке.** Базисы **(e_1, e_2)** и **(e_2, e_1)** - это два совершенно разных базиса.

Говоря простым языком, базис - это своеобразная “измерительная рулетка” для оценки нашего линейного пространства. Базис обычно принято описывать при помощи Декартовой системы координат XУ. Мы можем изобразить базисы для двухмерного и трехмерного пространства следующим образом:

$$S_2 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \qquad S_3 = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

Базис

У нас 2 базисных набора векторов: $S_2 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$ $T = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -3 \\ 1 \end{bmatrix} \right\}$

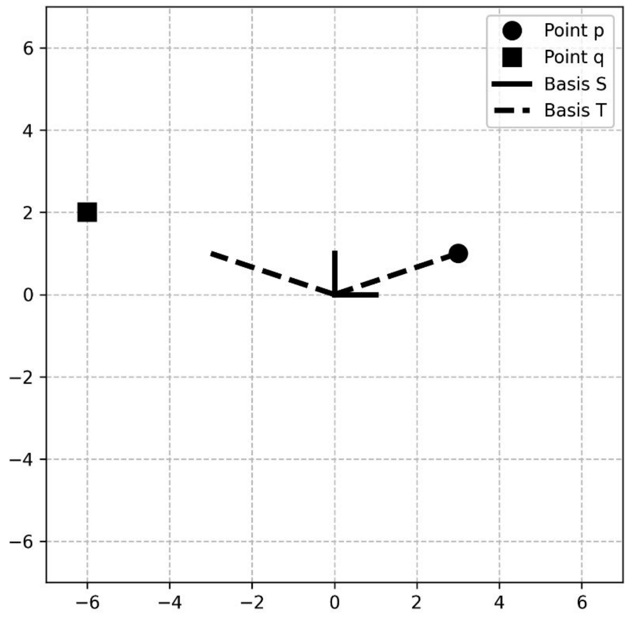
которые образуют подпространство **всех комбинаций векторов в $\mathbb{R}^2$**

Представьте, что мы хотим **выразить вектора p и q**, являющиеся нашими сэмплами из набора данных.

Мы можем выразить эти сэмплы как **их отношение к началу Декартовой системы координат** используя базис $\mathbf{S_2}$ или **T**

Координаты ***p = (3, 1)*** и ***q = (-6, 2)*** необходимо представить наши наблюдения в виде линейной комбинации базисных векторов.

Как это будет выглядеть?



Базис

У нас 2 базисных набора векторов: $S_2 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \quad T = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -3 \\ 1 \end{bmatrix} \right\}$

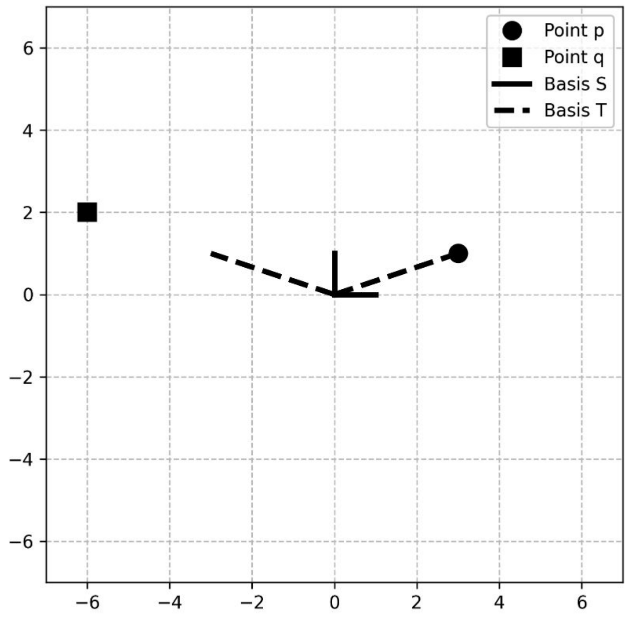
которые образуют подпространство **всех комбинаций векторов в $\mathbb{R}^2$**

Представьте, что мы хотим **выразить вектора p и q**, являющиеся нашими сэмплами из набора данных.

Мы можем выразить эти сэмплы как **их отношение к началу Декартовой системы координат** используя базис $\mathbf{S_2}$ или **T**

Координаты ***p = (3, 1)*** и ***q = (-6, 2)*** необходимо представить наши наблюдения в виде линейной комбинации базисных векторов.

Как это будет выглядеть в базисе S?



Базис

У нас 2 базисных набора векторов: $S_2 = \left\{ \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix} \right\} \quad T = \left\{ \begin{bmatrix}3\\1\end{bmatrix}, \begin{bmatrix}-3\\1\end{bmatrix} \right\}$

которые образуют подпространство **всех комбинаций векторов в $\mathbb{R}^2$**

Представьте, что мы хотим **выразить вектора p и q**, являющиеся нашими сэмплами из набора данных.

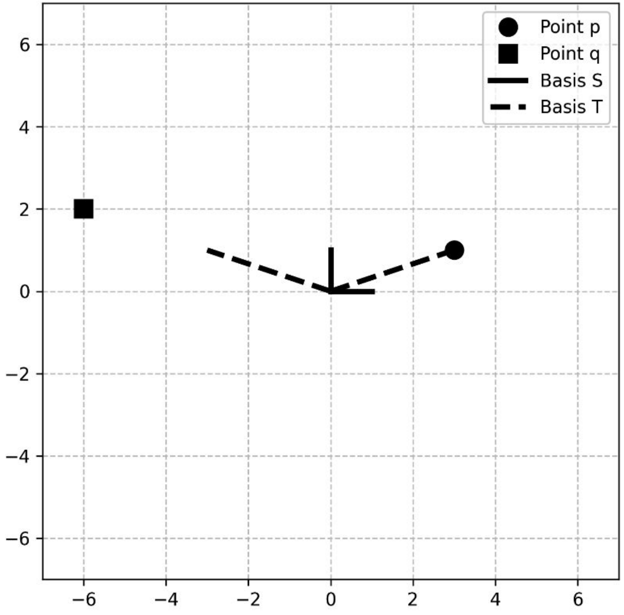
Мы можем выразить эти сэмплы как **их отношение к началу Декартовой системы координат** используя базис $\mathbf{S_2}$ или **T**

Координаты ***p = (3, 1)*** и ***q = (-6, 2)*** необходимо представить наши наблюдения в виде линейной комбинации базисных векторов.

Все довольно просто, если выразить наши наблюдения как линейную комбинацию базисных векторов, то получится следующее:

- **p**: $3s_1 + 1s_2$
- **q**: $-6s_1 + 2s_2$

А как это будет выглядеть через T?



Базис

У нас 2 базисных набора векторов: $S_2 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \quad T = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -3 \\ 1 \end{bmatrix} \right\}$

которые образуют подпространство **всех комбинаций векторов в $\mathbb{R}^2$**

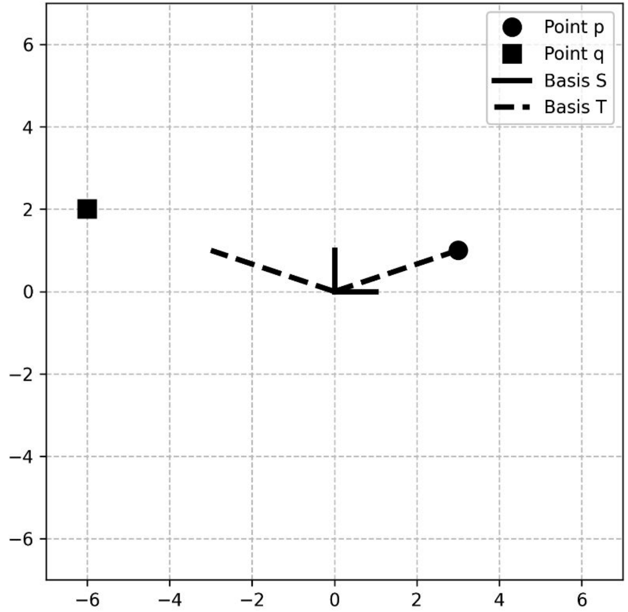
Представьте, что мы хотим **выразить вектора p и q**, являющиеся нашими сэмплами из набора данных.

Мы можем выразить эти сэмплы как **их отношение к началу Декартовой системы координат** используя базис **S_2** или **T**

Координаты в **базисе S**: ***p = (3, 1)*** и ***q = (-6, 2)*** необходимо представить наши наблюдения в виде линейной комбинации базисных векторов.

Все довольно просто, если выразить наши наблюдения как линейную комбинацию базисных векторов, то получится следующее:

- **p**: $3s_1 + 1s_2$
- **q**: $-6s_1 + 2s_2$



Базис

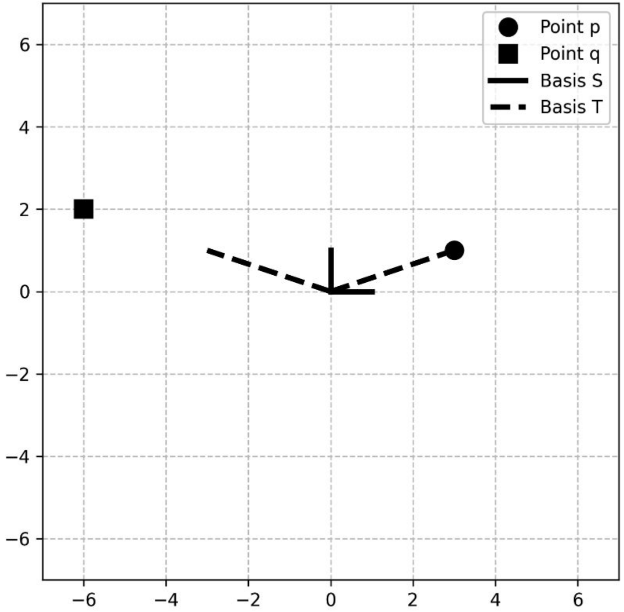
У нас 2 базисных набора векторов: $S_2 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \quad T = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -3 \\ 1 \end{bmatrix} \right\}$

которые образуют подпространство **всех комбинаций векторов в $\mathbb{R}^2$**

Представьте, что мы хотим **выразить вектора p и q**, являющиеся нашими сэмплами из набора данных.

Мы можем выразить эти сэмплы как **их отношение к началу Декартовой системы координат** используя базис $\mathbf{S_2}$ или **T**

Координаты в **базисе T**: ***p = (1, 0)*** и ***q = (0, 2)*** необходимо представить наши наблюдения в виде линейной комбинации базисных векторов.



Базис

У нас 2 базисных набора векторов: $S_2 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \quad T = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -3 \\ 1 \end{bmatrix} \right\}$

которые образуют подпространство **всех комбинаций векторов в $\mathbb{R}^2$**

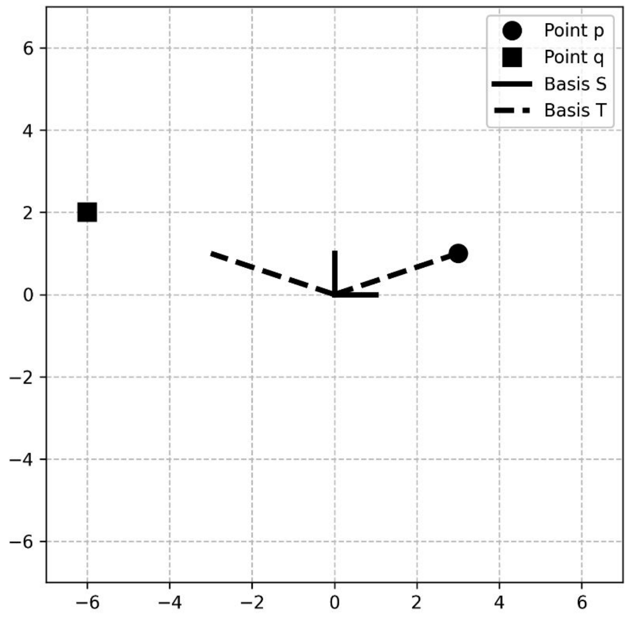
Представьте, что мы хотим **выразить вектора p и q**, являющиеся нашими сэмплами из набора данных.

Мы можем выразить эти сэмплы как **их отношение к началу Декартовой системы координат** используя базис **S_2** или **T**

Координаты в **базисе T:** ***p = (1, 0)*** и ***q = (0, 2)*** необходимо представить наши наблюдения в виде линейной комбинации базисных векторов.

- ***p***: $1t_1 + 0t_2 = t_1$
- ***q***: $0t_1 + 2t_2 = 2t_2$

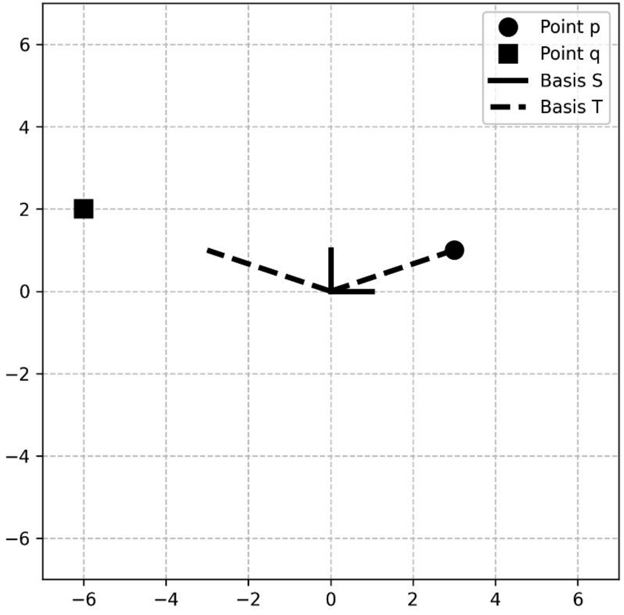
Наши наблюдения p и q остались такими же, не смотря на смену базиса, но в случае базиса T мы имеем более компактную ортогональную запись, что поможет нам в вычислениях.



Базис

Базис очень важная вещь в машинном обучении.

Многие проблемы в линейной алгебре могут быть выражены как задача по поиску лучшего базиса для описания некоторого линейного подпространства.



Это важно понимать, так как мы с ним встретимся еще в:

- алгоритмах понижения размерности
- поиске важных признаков для алгоритма
- анализе главных компонент
- факторном анализе
- в алгоритме SVD (singular value decomposition)
- линейном дискриминантном анализе
- аппроксимации изображений
- сжатии данных изображений
- полносвязной сверточной операции в сверточной нейронной сети

Применение вышесказанного в ML

Для того, чтобы вы не взаимодействовали долго с абстрактной информацией, которую мы разобрали выше, давайте подумаем где мы можем применять всё то, что выучили?

Применение вышесказанного в ML

Для того, чтобы вы не взаимодействовали долго с абстрактной информацией, которую мы разобрали выше, давайте подумаем где мы можем применять всё то, что выучили:

- Корреляция и косинусная мера (мера похожести)
- Фильтрация Time-Series данных и определения важности признаков
- k-Means кластеризация

И огромное количество других алгоритмов, операции которых выражены в линейном пространстве.

Применение вышесказанного в ML

Для того, чтобы вы не взаимодействовали долго с абстрактной информацией, которую мы разобрали выше, давайте подумаем где мы можем применять всё то, что выучили:

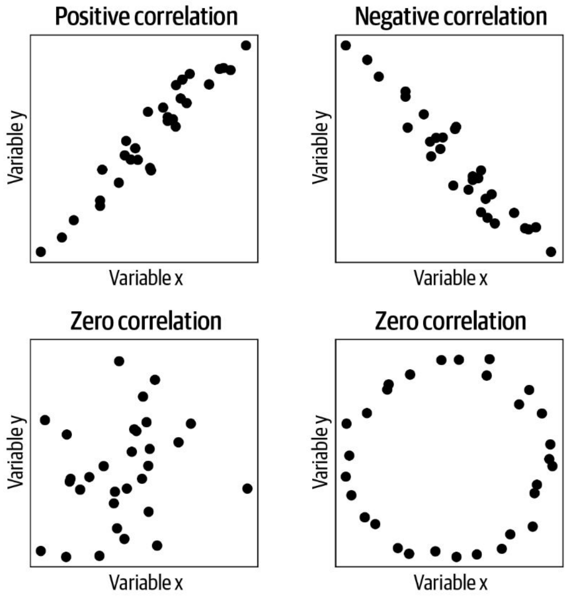
- Корреляция и косинусная мера (мера похожести)
- Фильтрация Time-Series данных и определения важности признаков
- k-Means кластеризация

И огромное количество других алгоритмов, операции которых выражены в линейном пространстве.

Корреляция

Корреляция (correlation) - одна из фундаментальных и важных метрик в статистическом анализе и машинном обучении.

Коэффициент корреляции - это скаляр, который показывает линейное **отношение между 2-мя переменными** (признаками). Коэффициент корреляции **принимает значения от -1 до +1**, показывая тем самым сильную позитивную взаимосвязь между признаками (сильную негативную, позитивную и отсутствие взаимосвязи).



Корреляция

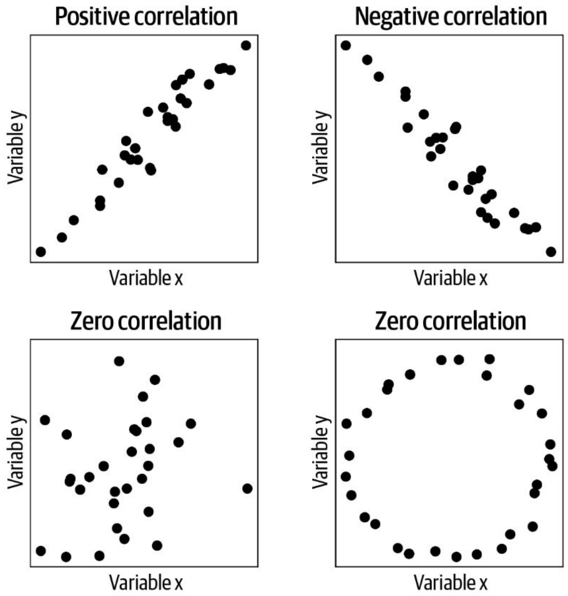
Корреляция (correlation) - одна из фундаментальных и важных метрик в статистическом анализе и машинном обучении.

Коэффициент корреляции - это скаляр, который показывает линейное **отношение между 2-мя переменными** (признаками). Коэффициент корреляции **принимает значения от -1 до +1**, показывая тем самым сильную позитивную взаимосвязь между признаками (сильную негативную, позитивную и отсутствие взаимосвязи).

Мы с вами помним, что скалярное произведение одна из основных операций в расчете коэффициента корреляции и величина скалярного произведения связана с величиной (масштабом) признаков. Таким образом нам надо добавить еще некую нормализацию для того чтобы коэффициент принимал значения от -1 до +1.

2 основных метода нормализации:

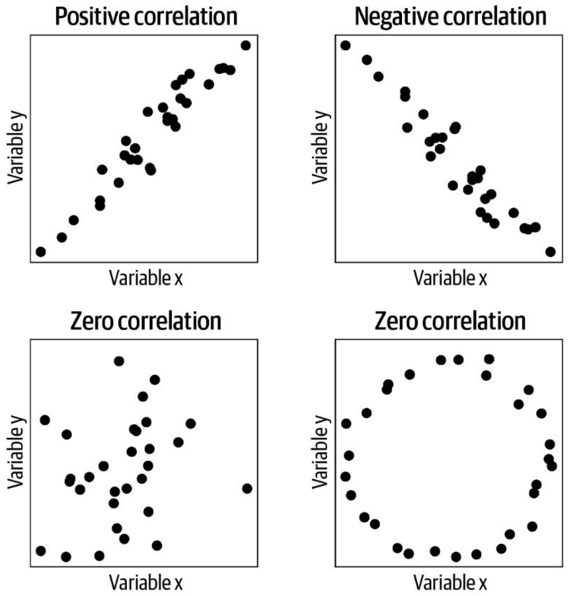
- Нормализация по среднему значению
- Деление скалярного произведения на скалярное произведение норм векторов



Корреляция

Коэффициент корреляции Пирсона:

$$\rho = \frac{\sum_{i=1}^{n}\left(x_i - \overline{x}\right)\left(y_i - \overline{y}\right)}{\sqrt{\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2}\sqrt{\sum_{i=1}^{n}\left(y_i - \overline{y}\right)^2}}$$



В формате векторных операций это будет выглядит следующим образом:

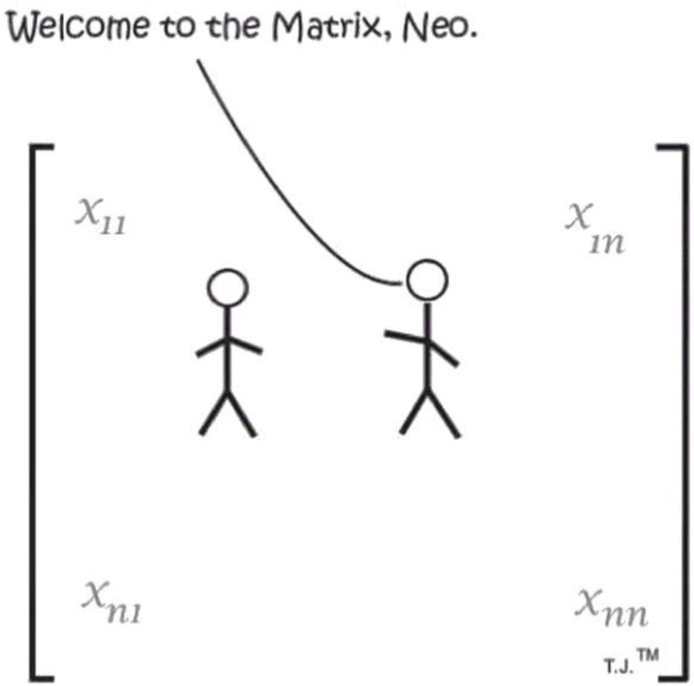
$$\rho = \frac{\widetilde{\mathbf{x}}^{\mathrm{T}}\widetilde{\mathbf{y}}}{\| \widetilde{\mathbf{x}} \| \| \widetilde{\mathbf{y}} \|}$$

Корреляция далеко не единственный метод для сравнения взаимосвязи 2-х признаков. Другой метод известен как **косинусная мера** или **косинусное расстояние**.

$$\cos\left(\theta_{x,y}\right) = \frac{\alpha}{\| \mathbf{x} \| \| \mathbf{y} \|}$$

α - скалярное произведение между x и y

B Jupyter Notebook


Welcome to the Matrix, Neo.
x_{11}
x_{1n}
x_{n1}
x_{nn}
T.J.™

Матрица

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

Матрицей размера m x n называется упорядоченная прямоугольная таблица (массив) чисел, содержащая m строк и n столбцов. Обозначается M_{mn}

Создание и визуализация матриц в NumPy

В зависимости от контекста задачи, матрицы используются в машинном обучении как **набор векторов-столбцов** (например датасет в разрезе различных признаков), как **набор векторов-строк** (например мультиканальные данные временного ряда) или как **тензор** (например картинки).

$$\begin{bmatrix} 1 & 2 \\ \pi & 4 \\ 6 & 7 \end{bmatrix}, \quad \begin{bmatrix} -6 & 1/3 \\ e^{4.3} & -1.4 \\ 6/5 & 0 \end{bmatrix}$$

Пример создания матрицы случайных чисел при помощи NumPy:

```
A = np.arange(60).reshape(6,10)
```

Как и любой итерируемый объект, мы можем обращаться к индексации:

```
sub = A[1:4:1,0:5:1]
```

Создание и визуализация матриц в NumPy

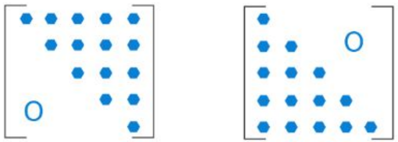
Создание **случайной** матрицы при помощи NumPy:

```
Mrows = 4 # shape [0]
Ncols = 6 # shape [1]
A = np.random.randn(Mrows,Ncols)
```

Диагональная матрица: `np.diag()`

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Треугольная матрица содержит нулевые элементы либо выше (верхнетреугольная), либо ниже главной диагонали (нижнетреугольная). `np.triu()` для верхнетреугольной (**tri**angular **u**pper) `np.tril()` для нижнетреугольной (**tri**angular **l**ower)



Единичная матрица - одна из наиболее важных матриц. Это **эквивалент единицы**, в том смысле, что любая матрица или вектор, умноженные на единичную матрицу, являются той же матрицей или вектором, что и в начале. Единичная матрица представляет с собой **квадратную диагональную матрицу, элементы главной диагонали которой равны 1.** Обозначается как **I** `np.eye()`

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2 x 2   3 x 3

Нулевая матрица - элементы которой равны 0 - `np.zeros()`

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Матрица и операции над матрицами

Операции над матрицами:

1. Сложение
2. Умножение на скаляр
3. Транспонирование матрицы
4. Умножение матриц
5. Взятие обратной матрицы

Пройдемся по всем основным операциям.

Операции над матрицами. Сложение/Вычитание.

$$A = \begin{bmatrix} 2 & 1 & 2 & 0 \\ 1 & 0 & 9 & 11 \\ 4 & 1 & 7 & 6 \\ 4 & 3 & 5 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 9 & -9 & 1 & 1 \\ 12 & -1 & -4 & 1 \\ 4 & -2 & -3 & 3 \\ 1 & 0 & 8 & 0 \end{bmatrix}$$

Сложение (вычитание) матриц интуитивная операция, которая выполняется поэлементно только для матриц одинаковых размерностей.

Операции над матрицами. Умножение на скаляр.

$$A = \begin{bmatrix} 2 & 1 & 2 & 0 \\ 1 & 0 & 9 & 11 \\ 4 & 1 & 7 & 6 \\ 4 & 3 & 5 & 0 \end{bmatrix} \qquad \lambda = 3$$

Умножение на скаляр выполняется также поэлементно. Каждый элемент матрицы умножается на скаляр. *Можно ли сложить матрицу и скаляр?*

Поэлементное умножение матриц

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \odot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 2a & 3b \\ 4c & 5d \end{bmatrix}$$

Поэлементное умножение матриц одинаковой размерности, где каждый элемент 1-й матрицы умножается на соответствующий ей элемент 2-й матрицы (известное также как умножение Адамара). В NumPy реализован через метод `np.multiply()`

```
A = np.random.randn(3,4)
B = np.random.randn(3,4)

A*B # Hadamard multiplication
np.multiply(A,B) # also Hadamard
A@B # NOT Hadamard!
```

Операции над матрицами. Транспонирование.

$$A \begin{bmatrix} 1 & 4 & 3 \\ 8 & 2 & 6 \\ 7 & 8 & 3 \\ 4 & 9 & 6 \\ 7 & 8 & 1 \end{bmatrix} \qquad A^{T} \begin{bmatrix} 1 & 8 & 7 & 4 & 7 \\ 4 & 2 & 8 & 9 & 8 \\ 3 & 6 & 3 & 6 & 1 \end{bmatrix}$$

Транспонированием матрицы называется операция, в результате которой образуется новая матрица, где строками служат столбцы исходной, записанные с сохранением их следования `np.transpose()` либо при помощи вызова атрибута объекта матрицы NumPy `matrix.T`

Операции над матрицами. Умножение матриц.

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

Чтобы умножить матрицу А и В необходимо чтобы внутренние размерность этих матриц совпадали, то есть

$$A_{3\ 3} * B_{3\ 3}$$

Размерность результирующей матрицы равна внешней размерности умножаемых матриц, т.е. C_{3x3}

Интерпретация матричного умножения в ML

Как вы помните, результат скалярного произведения - это скаляр, который показывает взаимосвязь между 2-мя векторами.

Результатом умножения матриц является матрица, в которой хранятся все попарные линейные связи между строками левой матрицами и столбцами правой матрицы.

Это важная вещь в машинном обучении, которая является основой для вычисления ковариоционных и корреляционных матриц, общей линейной модели, сингулярного разложения матриц и бесчисленного множества прочих подходов в машинном обучении и статистическом анализе данных.

Операции над матрицами. Умножение матриц.

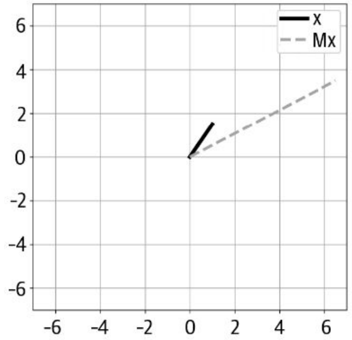
$$A_{33} = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 0 & 9 \\ 4 & 1 & 7 \end{bmatrix} \qquad B_{33} = \begin{bmatrix} 2 & 3 & 3 \\ 5 & 2 & 2 \\ 2 & 1 & 1 \end{bmatrix}$$

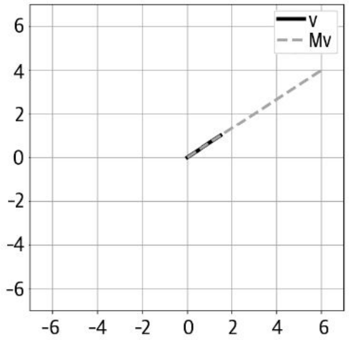
Геометрические трансформации

Когда мы представляем вектор как геометрический объект (линия), то умножение матрицы на вектор становится способом вращения и масштабирования этого вектора (умножение скаляра на вектор выполняет только операцию масштабирования).

Представим себе двумерное пространство и у нас есть матрица и соответствующие вектора:

```
M  = np.array([ [2,3],[2,1] ])
x  = np.array([ [1,1.5] ]).T
v  = np.array( [1.5, 1] )
Mx = M@x
Mv = M@v
```





Таким образом умножение матрицы M на вектор x поворачивает наш вектор и растягивает его вдоль его направления.

Во второй операции (умножение на вектор v) только растянуло вектор в его изначальных координатах.

Норма матрицы

На самом деле такого понятия как **“норма матрицы” не существует** в линейной алгебре.

Но тем не менее можно вычислить несколько различных норм из матрицы. Матричные нормы в чем-то схожи с векторными нормами тем, что каждая норма дает одно число, которое характеризует матрицу.

Норма матрицы $\boldsymbol{A}$ обозначается как $\|\ \boldsymbol{A}\ \|$

Различные матричные нормы имеют разное значение. Общая группировка матричных норм:

- нормы вычисленные путем поэлементных операций (element-wise norms, entrywise norms)
- Индуцированные формы норм (induced)

Норма Фробениуса

Евклидова норма, она же норма Фробениуса рассчитывается как сумма квадратов всех элементов матрицы взятых из под корня:

$$\| \mathbf{A} \|_F = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij}^2}$$

Норма Фробениуса также известна как **ℓ2** норма.

Существует огромное количество различных норм, но самые применимые в машинном обучении - это **ℓ0, ℓ1, ℓ2** и **ℓ3 нормы**.

Общая формула расчета для p-нормы, где p - степень нормы выглядит следующим образом: $\| \mathbf{A} \|_p = \left(\sum_{i=1}^{M} \sum_{j=1}^{N} |a_{ij}|^p \right)^{1/p}$

Нормы очень часто мы будем применять в техниках регуляризации, цель которых улучшить качество модели и увеличить обобщающую силу модели для данных, которые не участвовали в обучении алгоритма. Базовая идея регуляризации это добавить матричную норму в качестве функции ошибки в алгоритм оптимизации.

Норма поможет нам предотвратить процесс бесконечного увеличения параметров модели (**ℓ2** норма или ridge regression) либо поощрять пространство признаков за разные параметры модели (**ℓ1** норма или lasso regression).

Операции над матрицами. Определитель матрицы.

Определитель матрицы 2x2

$$\begin{vmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{vmatrix}$$

Детерминантом (определителем) квадратной матрицы 2-го порядка называется число:

$$\det \begin{vmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{vmatrix} = \alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21}$$

Операции над матрицами. Определитель матрицы.

Определитель матрицы 3x3

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = aei + bfg + cdh - ceg - bdi - afh$$

Определитель матрицы 4x4

$$\begin{vmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{vmatrix} = \begin{matrix} afkp - aflo - agjp + agln + ahjo - ahkn - bekp + belo \\ +bgip - bglm - bhio + bhkm + cejp - celn - cfip + cflm \\ +chin - chjm - dejo + dekn + dfio - dfkm - dgin + dgjm \end{matrix}$$

Операции над матрицами. Определитель матрицы.

$$\det \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = 1 \times 4 - 2 \times 3 = 4 - 6 = -2$$

Вычислить определитель для следующих матриц:

$$\begin{vmatrix} -1 & -2 \\ 6 & 3 \end{vmatrix} \qquad \begin{vmatrix} 2 & -1 \\ 4 & 1 \end{vmatrix} \qquad \begin{vmatrix} -4 & 2 \\ -8 & x \end{vmatrix}$$

Операции над матрицами. Определитель матрицы.

2 важных свойства определителя:

- он определен только и только для квадратных матриц
- определитель равен 0 для сингулярных матриц (матриц пониженного ранга)

Как интерпретировать определитель матрицы?

Операции над матрицами. Определитель матрицы.

2 важных свойства определителя:

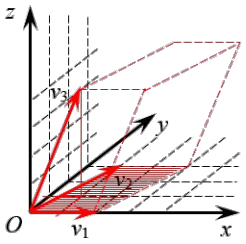
- он определен только и только для квадратных матриц
- определитель равен 0 для сингулярных матриц (матриц пониженного ранга)

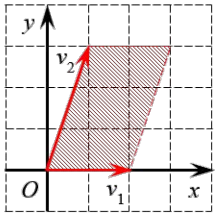
Как интерпретировать определитель матрицы:

Геометрическая интерпретация определителя связана с тем, насколько наша матрица “растягивает” наше векторное пространство при умножении матрицы на вектор (грубо говоря, показывает какую площадь или объем мы занимаем в нашем пространстве).

Определитель важен при расчете собственных векторов и при вычислении обратной матрицы, сингулярном разложении матрицы и других расчетах.

На практике определители матриц больших порядков считать проблематично и легко запутаться. В пакете NumPy вычисление определителя реализовано через метод `np.linalg.det()` и в пакете SciPy через метод `scipy.linalg.det()`





Операции над матрицами. Обратная матрица.

$$A A^{-1} = I$$

$$(A \mid I) \sim (I \mid A^{-1})$$

Вычислить обратную матрицу при помощи NumPy и проверьте что на выходе получается единичная матрица:

$$\begin{vmatrix} 2 & 3 \\ 3 & 5 \end{vmatrix} \qquad \begin{vmatrix} 3 & 1 & 0 \\ 2 & 4 & 5 \\ 0 & 2 & 3 \end{vmatrix}$$

Операции над матрицами. Базис.

Система векторов линейного пространства ***L*** образует базис в ***L*** если данная система векторов упорядочена, линейно независима и любой вектор из пространства ***L*** линейно выражается через векторы системы.

Являются ли вектора базисными:

$\bar{a}$ (-2, 1);
$\bar{e}$ (0, -2);

Операции над матрицами. Ранг.

Определение:

Ранг матрицы - наивысший порядок матрицы, отличный от нуля.

Минор k-го порядка матрицы - определитель квадратной матрицы kxk, которая составлена из элементов матрицы A, находящихся в заранее выбранных k-строках и k-столбцах, при этом сохраняется положение элементов матрицы A.

Простыми словами, если в матрице A вычеркнуть (p-k) строк и (n-k) столбцов, а из тех элементов, которые остались составить матрицу, сохраняя расположение элементов матрицы A, то определитель полученной матрицы и есть минор порядка k матрицы A.

В NumPy ранг матрицы вычисляется при помощи `np.linalg.matrix_rank()`

Операции над матрицами. Собственные числа/вектора.

Определение:

Пусть задана квадратная матрица $A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$

ненулевой вектор $X = \begin{vmatrix} x^1 \\ x_2 \\ x_3 \end{vmatrix}$ называется собственным вектором матрицы A,

если существует ненулевое число λ, такое что ***AX = λX***
Число λ при этом называется собственным значением вектора X относительно матрицы A.

Матрица A - λE называется характеристической матрицей матрицы A, многочлен |A - λE| называется характеристическим многочленом матрицы A, уравнение |A - λE| = 0 называется характеристическим уравнением матрицы A.

Операции над матрицами. Собственные числа/вектора.

Найти собственные числа/собственные вектора при помощи NumPy:

$$\begin{vmatrix} 3 & 4 \\ 5 & 2 \end{vmatrix} \qquad \begin{bmatrix} 5 & -7 & 7 \\ -8 & 3 & -2 \\ -8 & -1 & 2 \end{bmatrix} \qquad \begin{bmatrix} -7 & -8 & -8 \\ 4 & 5 & 4 \\ 2 & 4 & -1 \end{bmatrix}$$

Q&A

Дополнительные материалы

Теория и практика:

- **Книга** по математике для машинного обучения
- Тренировка по Linear Algebra. Задания от MIT: номер **1, 2, 3, 4, 5, 6, 7, 8, 9 и 10**
- Интерактивные **упражнения** от Khan Academy

Источники информации:

- Канал **3blue1brown**, раздел по основам линейной алгебры
- **Заметки** от курса университета Stanford по линейной алгебре
- **Мини-курс** по основам линейной алгебры для машинного обучения от Имперского Колледжа Лондона
- **Геометрическая интерпретация** операций над матрицами