

# Методы снижения размерности

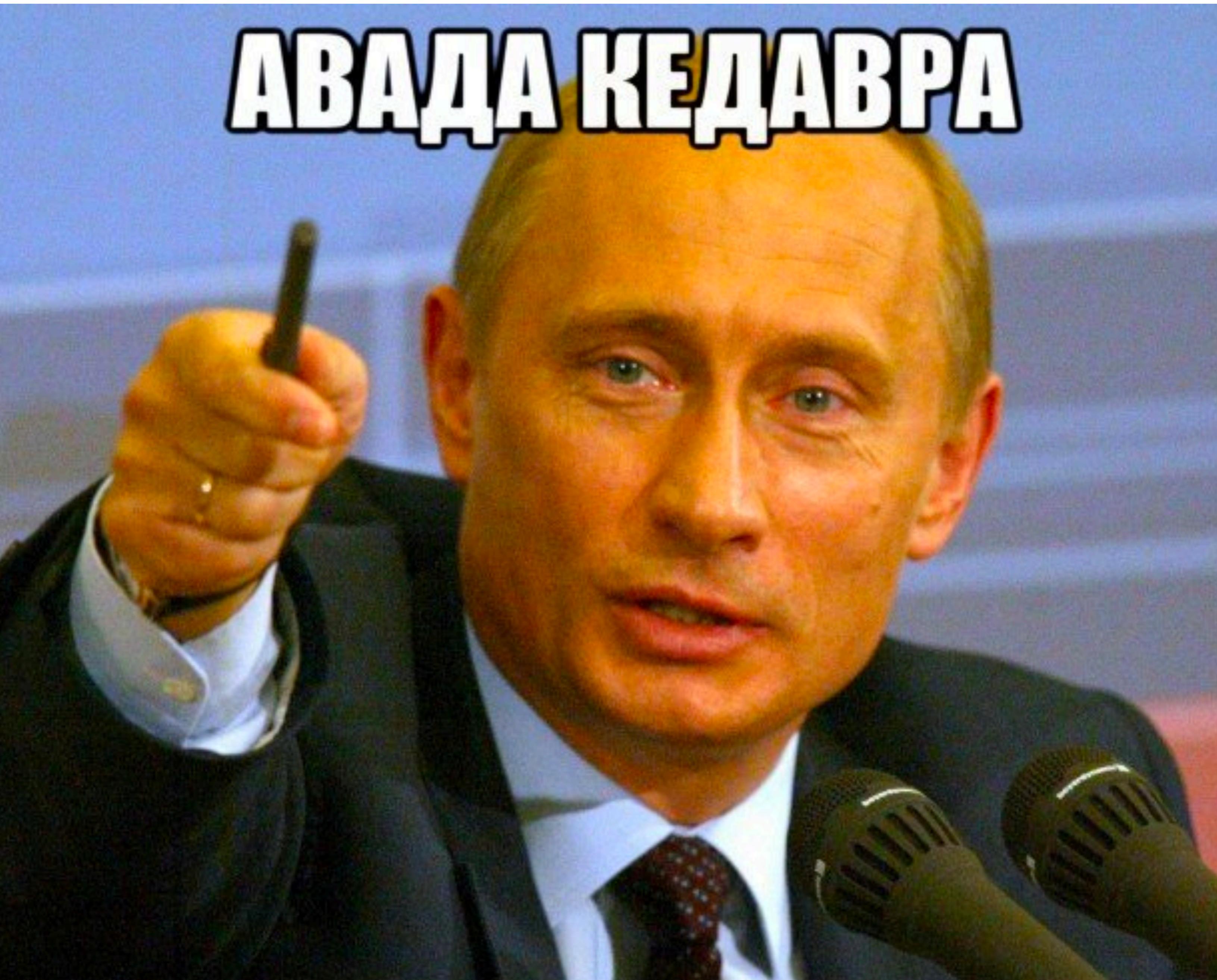
Парпулов Дмитрий  
программист в команде  
машиинного обучения почты



# Проклятие размерности

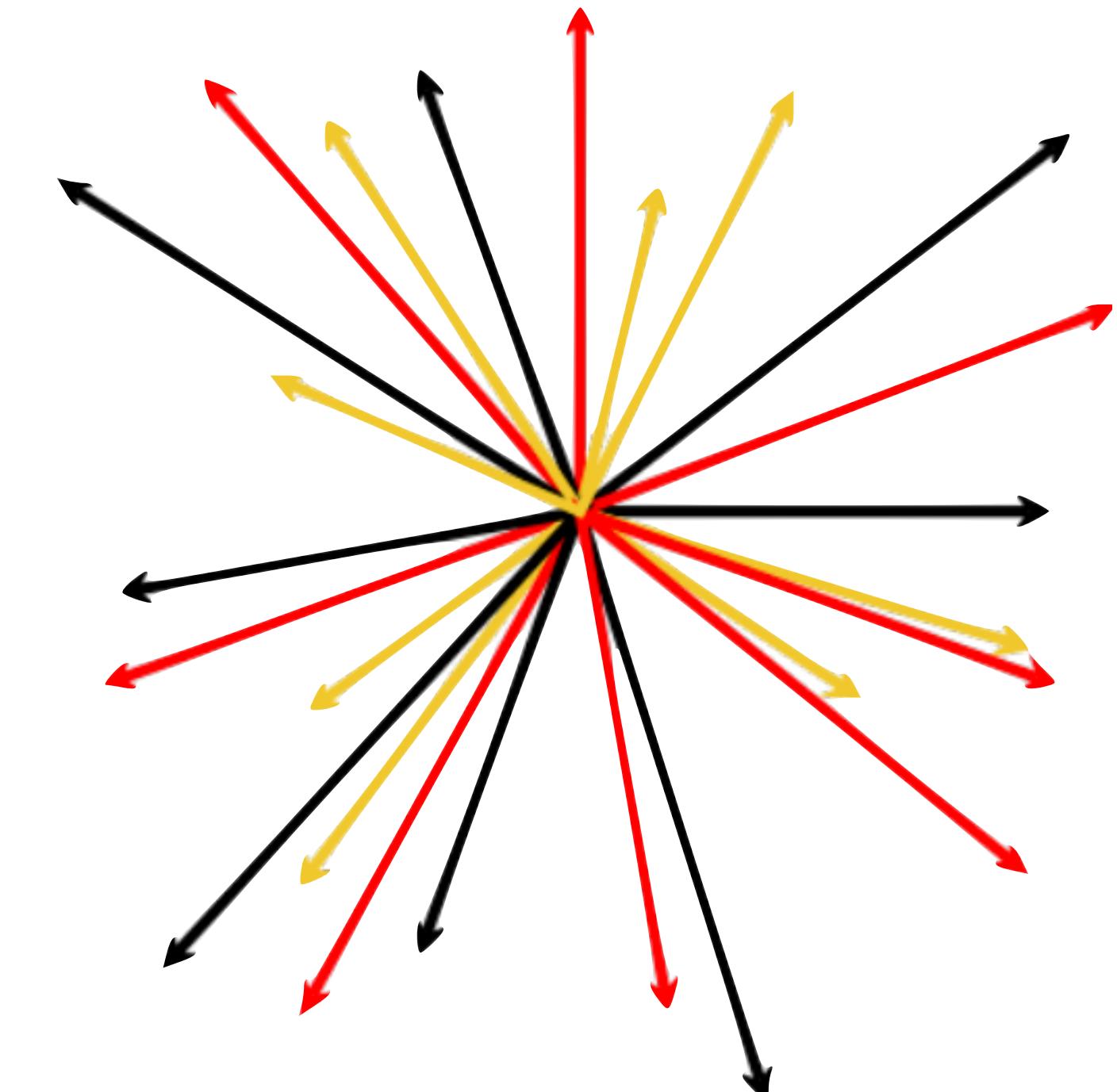


# АВАДА КЕДАВРА



# Проклятие размерности

- проблема, связанная с **экспоненциальным возрастанием количества данных из-за увеличения размерности пространства.**



Термин введен Р. Беллманом в 1961 году.

# Проклятие размерности

В чем идея?

$$x_1 = (a_1, a_2, \dots, a_N)$$

# Проклятие размерности

В чем идея?

$$x_1 = (a_1, a_2, \dots, a_N)$$

$$x_2 = (a_1 + \varepsilon, a_2 + \varepsilon, \dots, a_N + \varepsilon)$$

# Проклятие размерности

В чем идея?

$$x_1 = (a_1, a_2, \dots, a_N)$$

$$x_2 = (a_1 + \varepsilon, a_2 + \varepsilon, \dots, a_N + \varepsilon)$$

$$x_3 = (a_1, a_2 + \Delta, a_3, \dots, a_N)$$

# Проклятие размерности

В чем идея?

$$x_1 = (a_1, a_2, \dots, a_N)$$

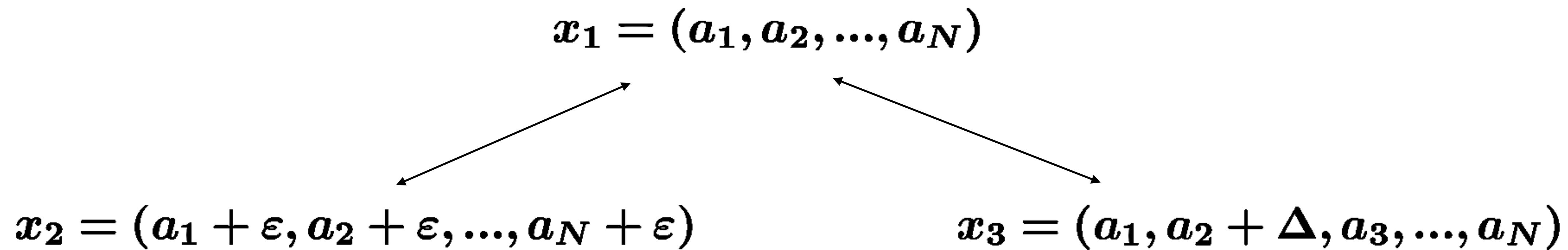
$$x_2 = (a_1 + \varepsilon, a_2 + \varepsilon, \dots, a_N + \varepsilon)$$

$$x_3 = (a_1, a_2 + \Delta, a_3, \dots, a_N)$$



# Проклятие размерности

В чем идея?



В высокомерном пространстве *небольшие* изменения в *большом* количестве координат вызывают сравнимые изменения в расстоянии, как и значительное изменение в 1 координате

# Проклятие размерности

При увеличении размерности пространства для равномерного покрытия многомерного пространства требуется **экспоненциальное увеличение количества данных**

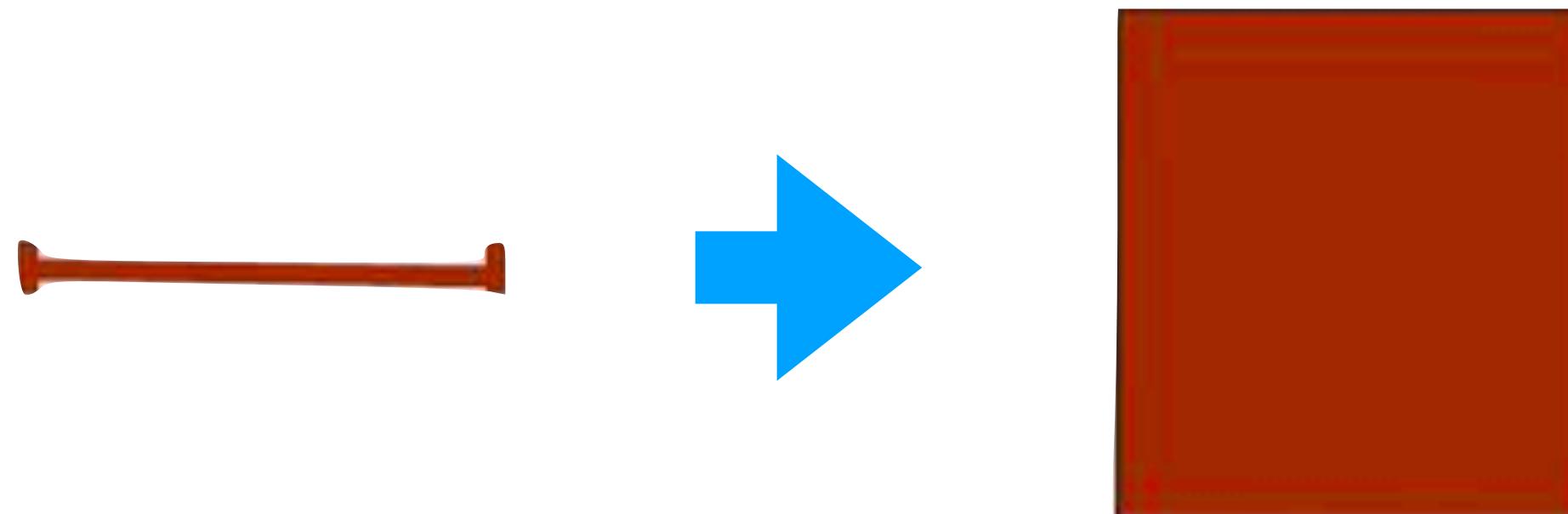
# Проклятие размерности

При увеличении размерности пространства для равномерного покрытия многомерного пространства требуется **экспоненциальное увеличение количества данных**



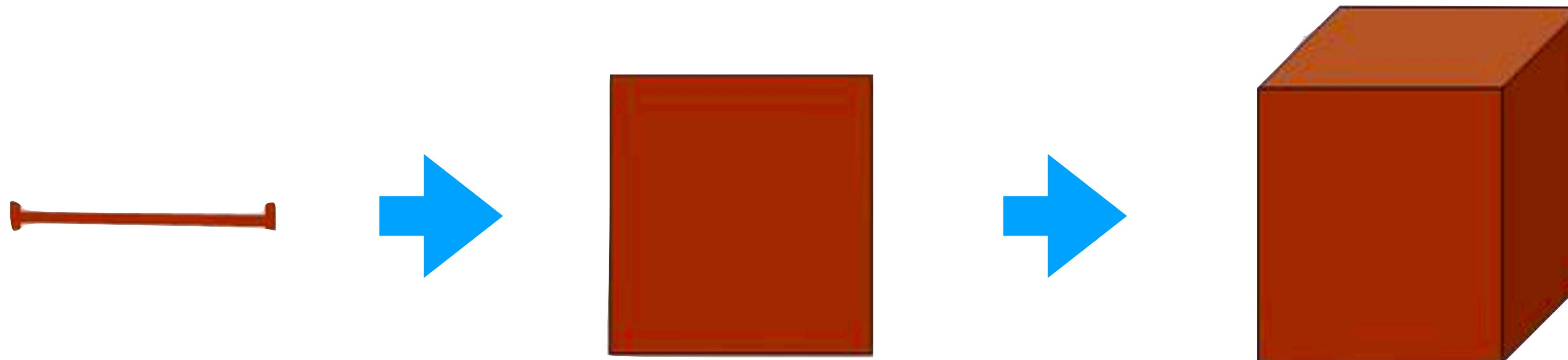
# Проклятие размерности

При увеличении размерности пространства для равномерного покрытия многомерного пространства требуется **экспоненциальное увеличение количества данных**



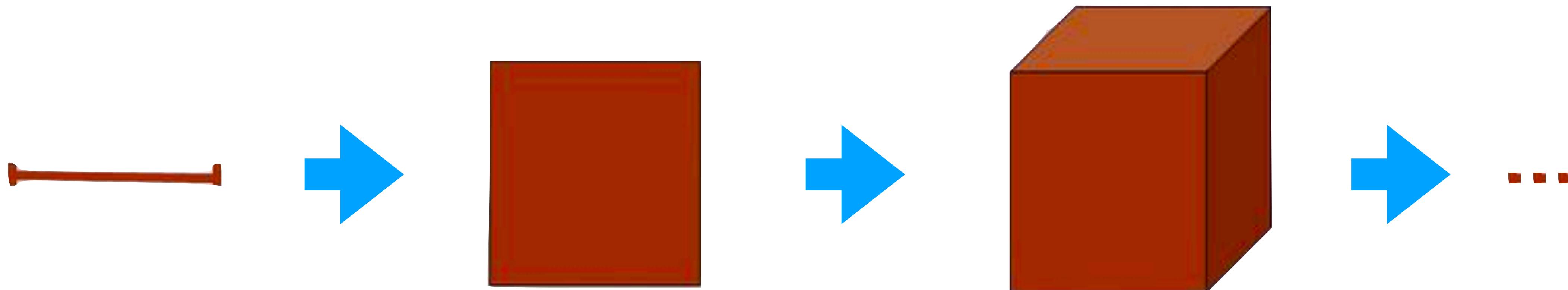
# Проклятие размерности

При увеличении размерности пространства для равномерного покрытия многомерного пространства требуется **экспоненциальное увеличение количества данных**



# Проклятие размерности

При увеличении размерности пространства для равномерного покрытия многомерного пространства требуется **экспоненциальное увеличение количества данных**



# Проклятие размерности

проявляется при работе со сложными системами, которые описываются большим числом параметров

- Трудоемкость вычислений
- Мультиколлинеарность и переобучение (лин.клф)
- Расстояния во всех парах объектов стремятся к одному и тому же значению -> становятся неинформативными (метрические клф)

# Как устранить проклятие размерности

- Снизить размерность пространства  
(перейти в подпространство меньшей  
размерности)
- Выполнить отбор признаков

# Отбор признаков



**МЫ ОТБИРАЕМ**

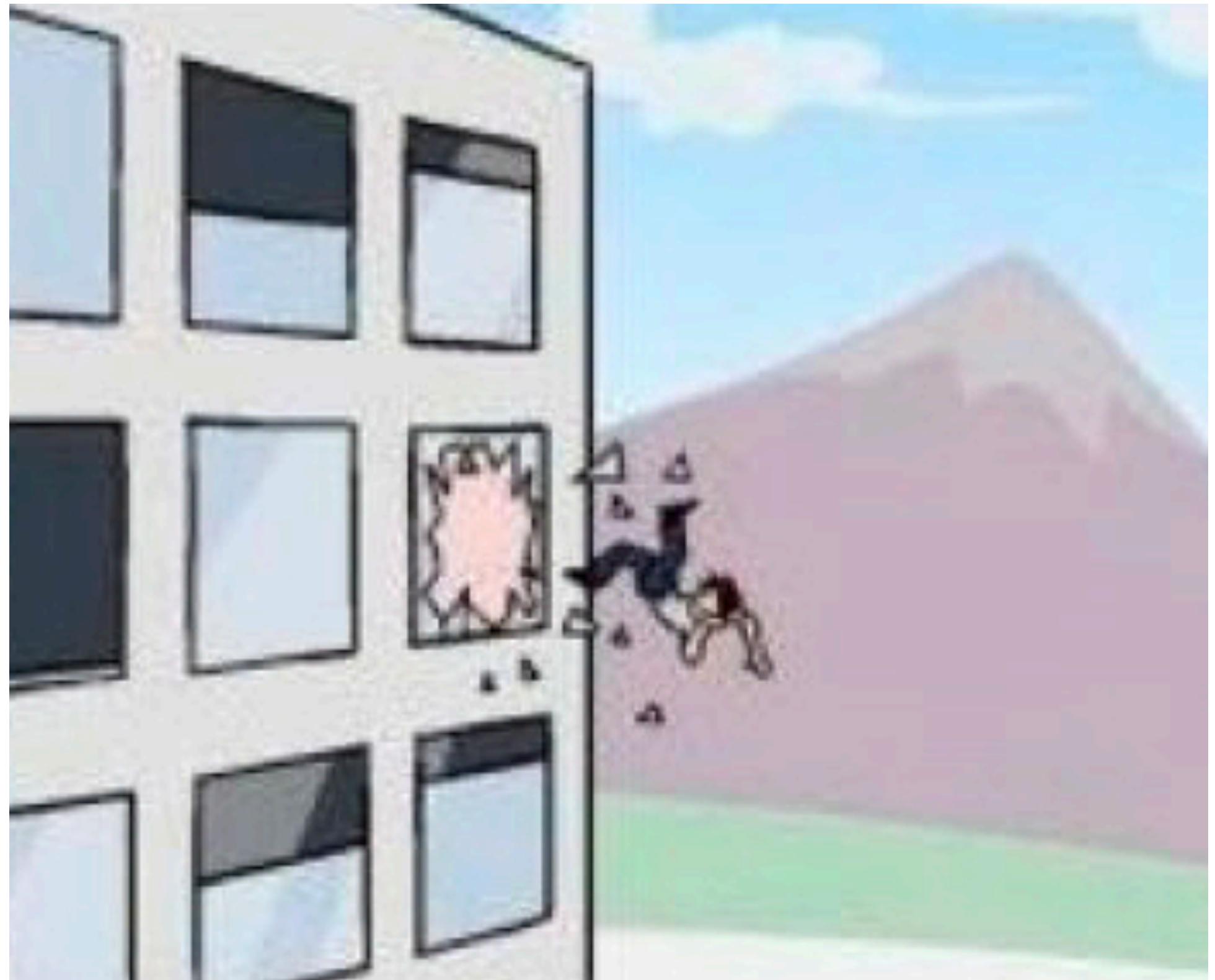
**ТОЛЬКО ЛУЧШИЕ ЗЕРНА**

# Зачем отбирать признаки

- данных мало, а признаков много
- могут быть шумовые признаки (бесполезные)
- признаки могут коррелировать
- вызывают переусложнение модели
- ограничение на ресурсы (модель/данные не влезают в память)

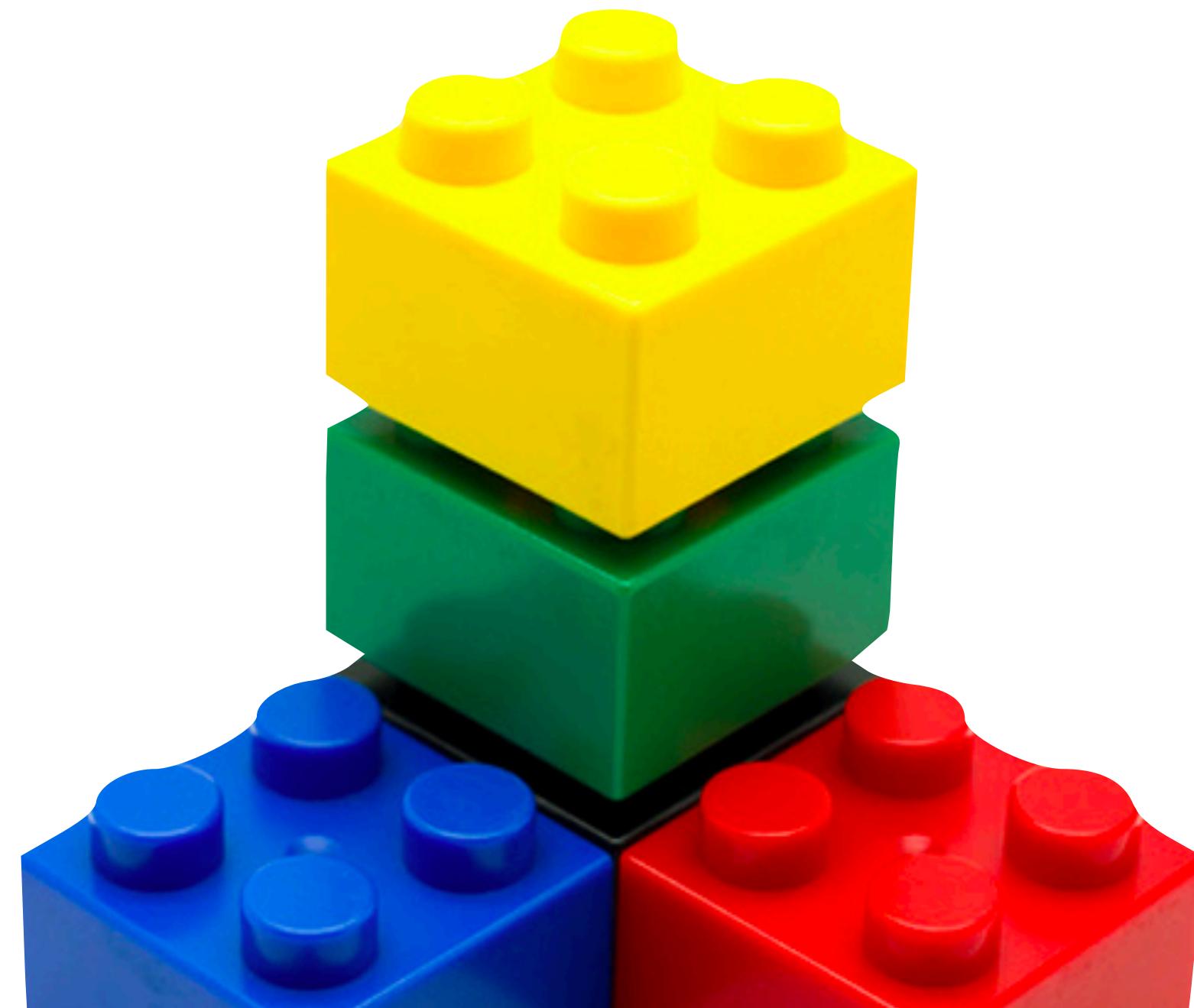
# Отбор признаков

- Можно выбросить часть признаков



# Отбор признаков

- Можно выбросить часть признаков
- Можно построить новые признаки на основе старых - снижение размерности



# Как отбирать признаки

- одномерный отбор
- жадные методы
- переборные методы
- на основе моделей

# Одномерный отбор

Оцениваем информативность каждого признака  
для решения задачи

# Одномерный отбор

Оцениваем информативность каждого признака  
для решения задачи

- Корреляция с таргетом
- PMI с таргетом (pointwise mutual information)
- Классификатор на 1 признаке

# Одномерный отбор

Оцениваем информативность каждого признака  
для решения задачи

- Корреляция с таргетом
- PMI с таргетом (pointwise mutual information)
- Классификатор на 1 признаке

Не учитываются сложные зависимости между  
признаками

# Жадные методы отбора признаков



# Жадные методы отбора признаков

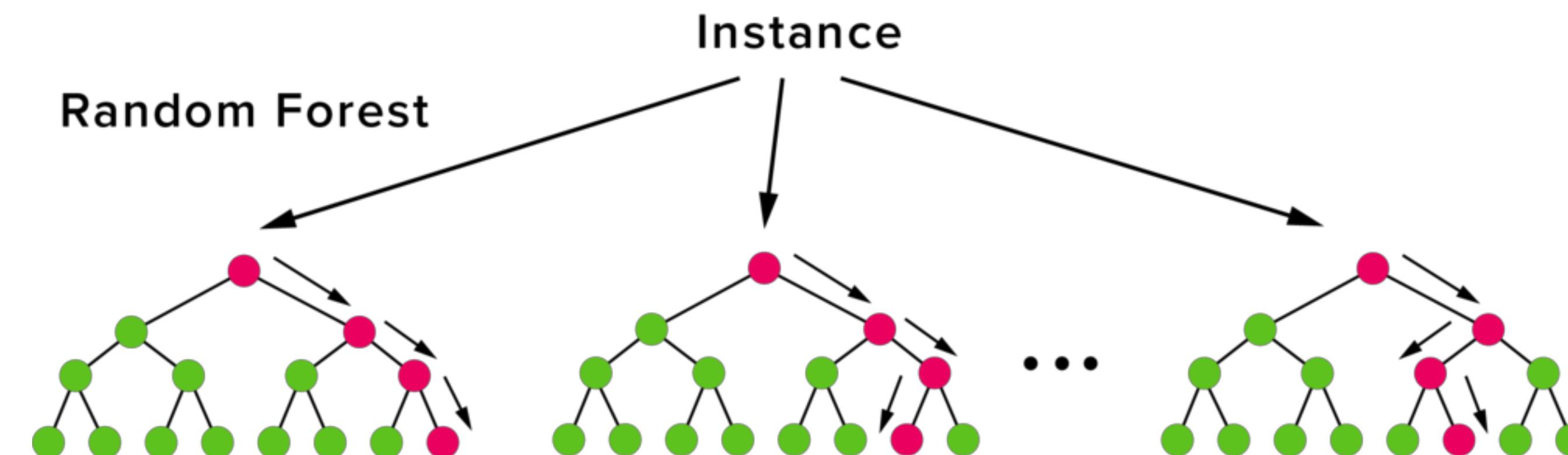
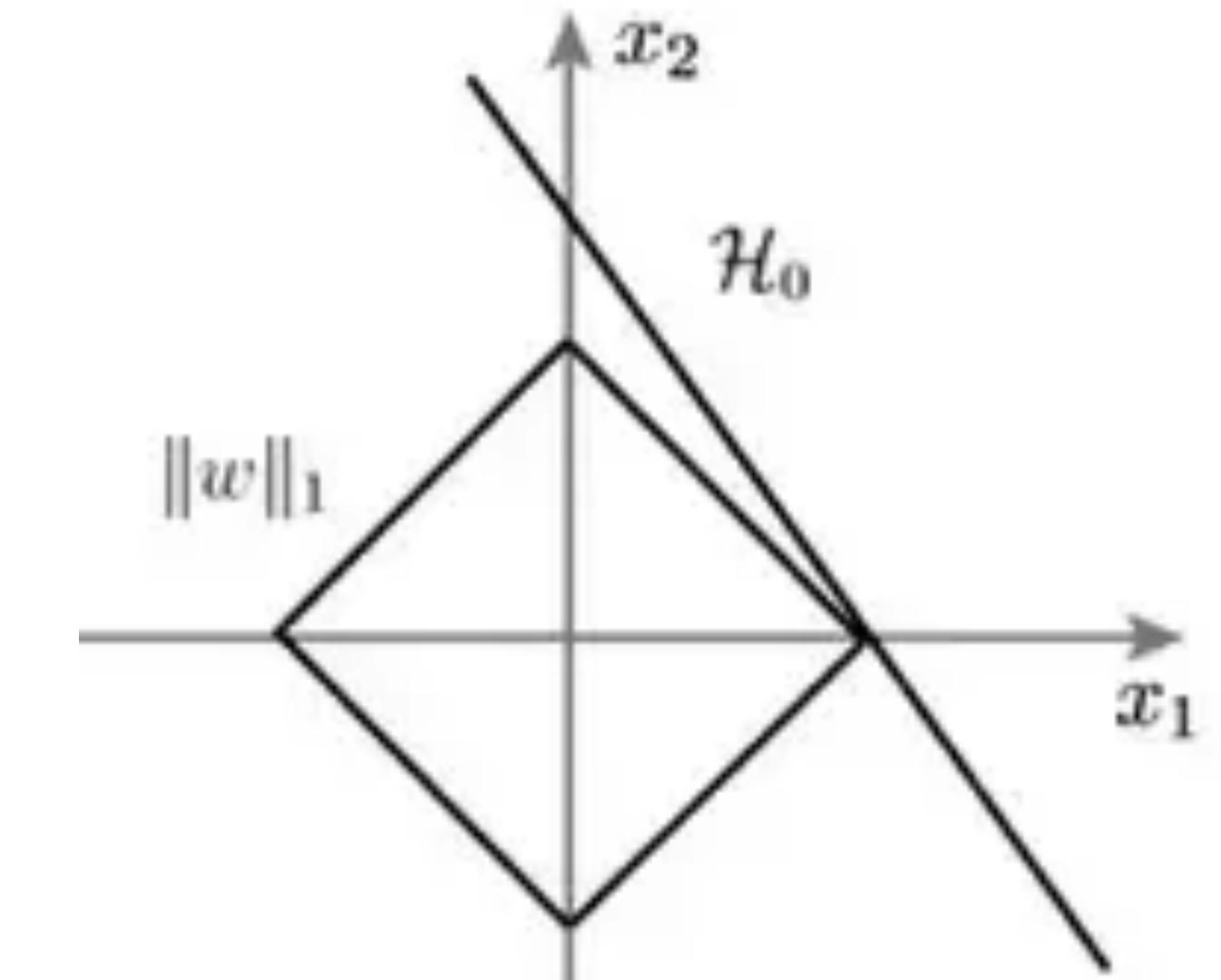
- Перебираем комбинации (полный перебор, добавление, удаление, ADD-DEL)
- Для каждой комбинации признаков обучаем модель
- Выбираем комбинацию с лучшим качеством

# Жадные методы отбора признаков

- Перебираем комбинации (полный перебор, добавление, удаление, ADD-DEL)
- Для каждой комбинации признаков обучаем модель
- Выбираем комбинацию с лучшим качеством
  - Много комбинаций
  - Жадные методы
  - Обучение модели на каждом наборе признаков

# Отбор на основе моделей

- Линейные модели
- $\ell_1$ -регуляризация
- Деревья, случайный лес, бустинг над деревьями



# Singular Value Decomposition

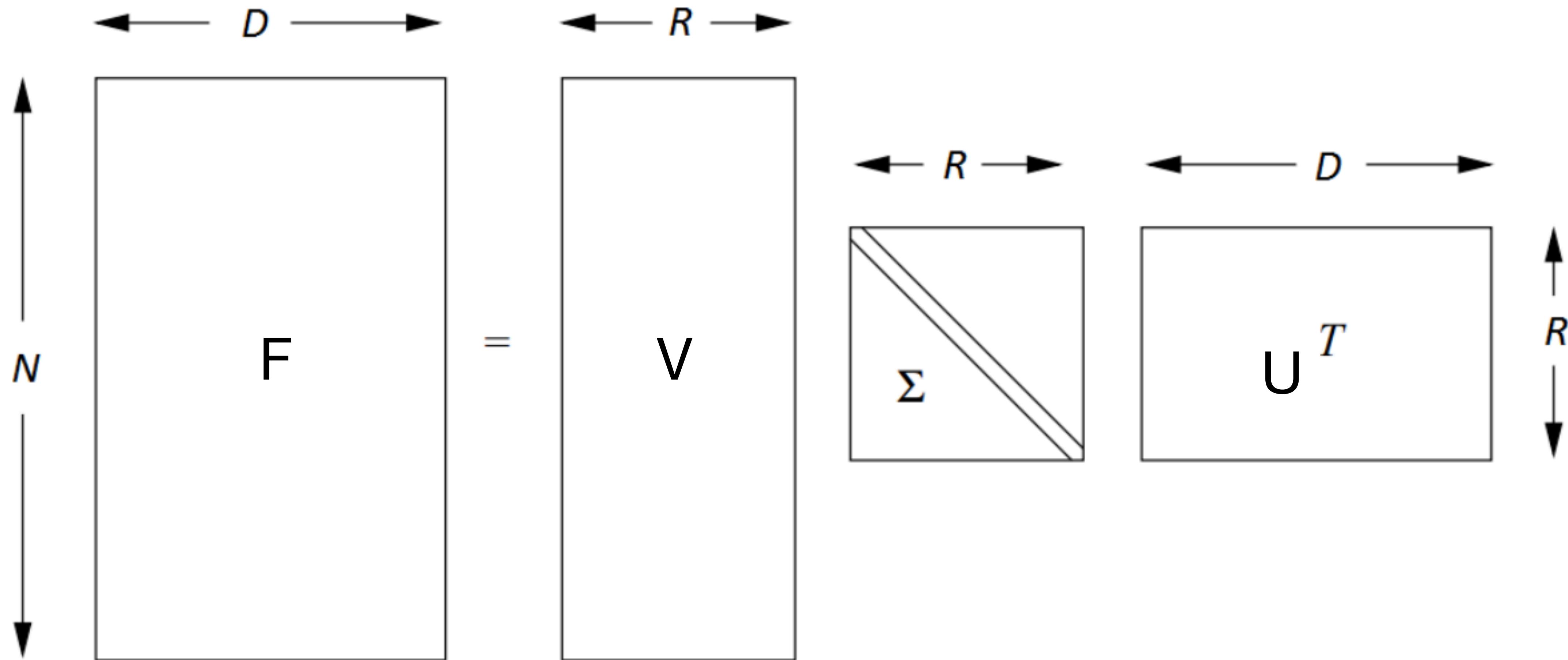
# Singular Value Decomposition

Любая вещественная матрица  $F$  разложима следующим образом:

$$F = V\Sigma U^T$$

- $U$  и  $V$  - ортогональные матрицы
- $\Sigma$  - диагональная

# Singular Value Decomposition



# SVD intuition

$$F = V\Sigma U^T$$

Пусть исходная матрица  $F$  задает некоторое линейное преобразование исходного пространства.

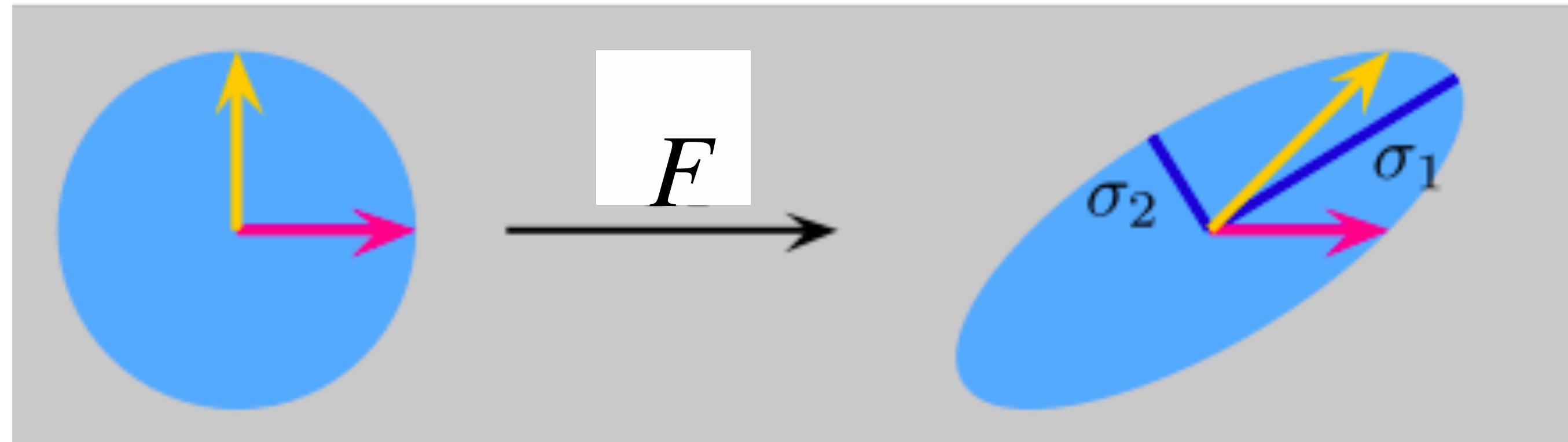
# SVD intuition

$$F = V\Sigma U^T$$

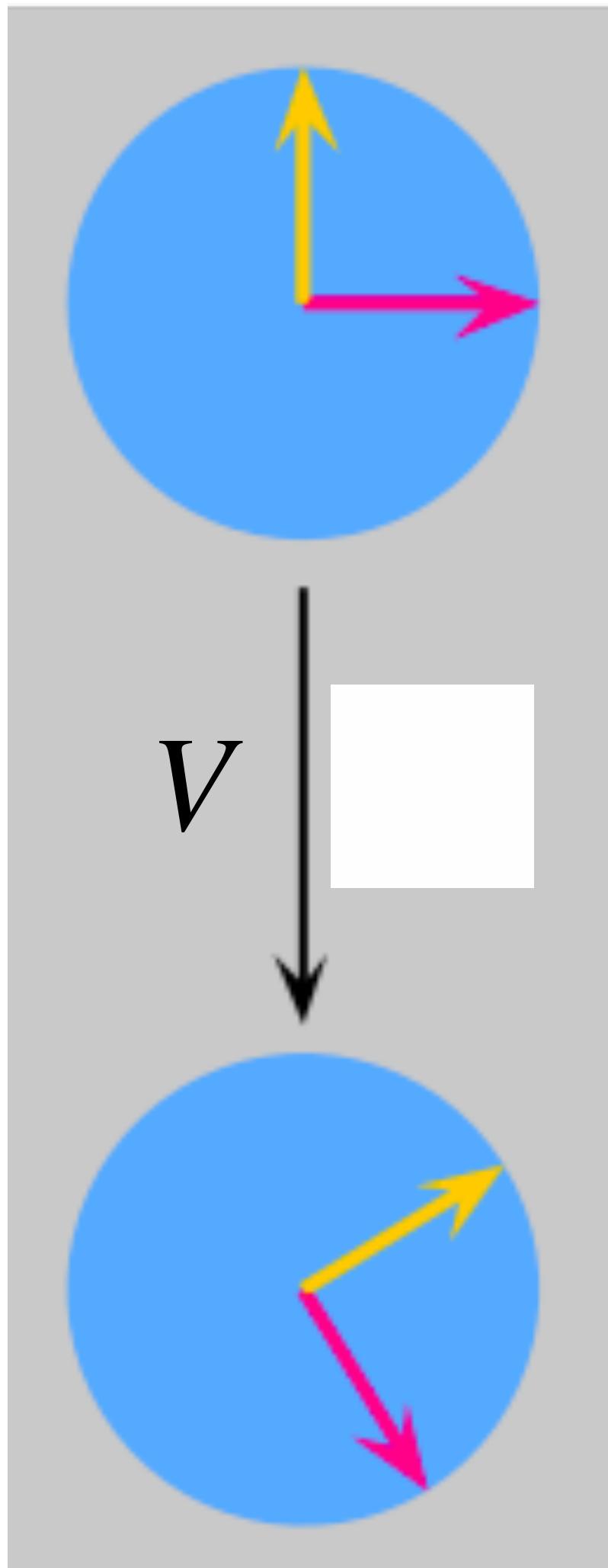
Пусть исходная матрица  $F$  задает некоторое линейное преобразование исходного пространства.

Тогда это линейное преобразование можно разбить на последовательность более простых линейных преобразований

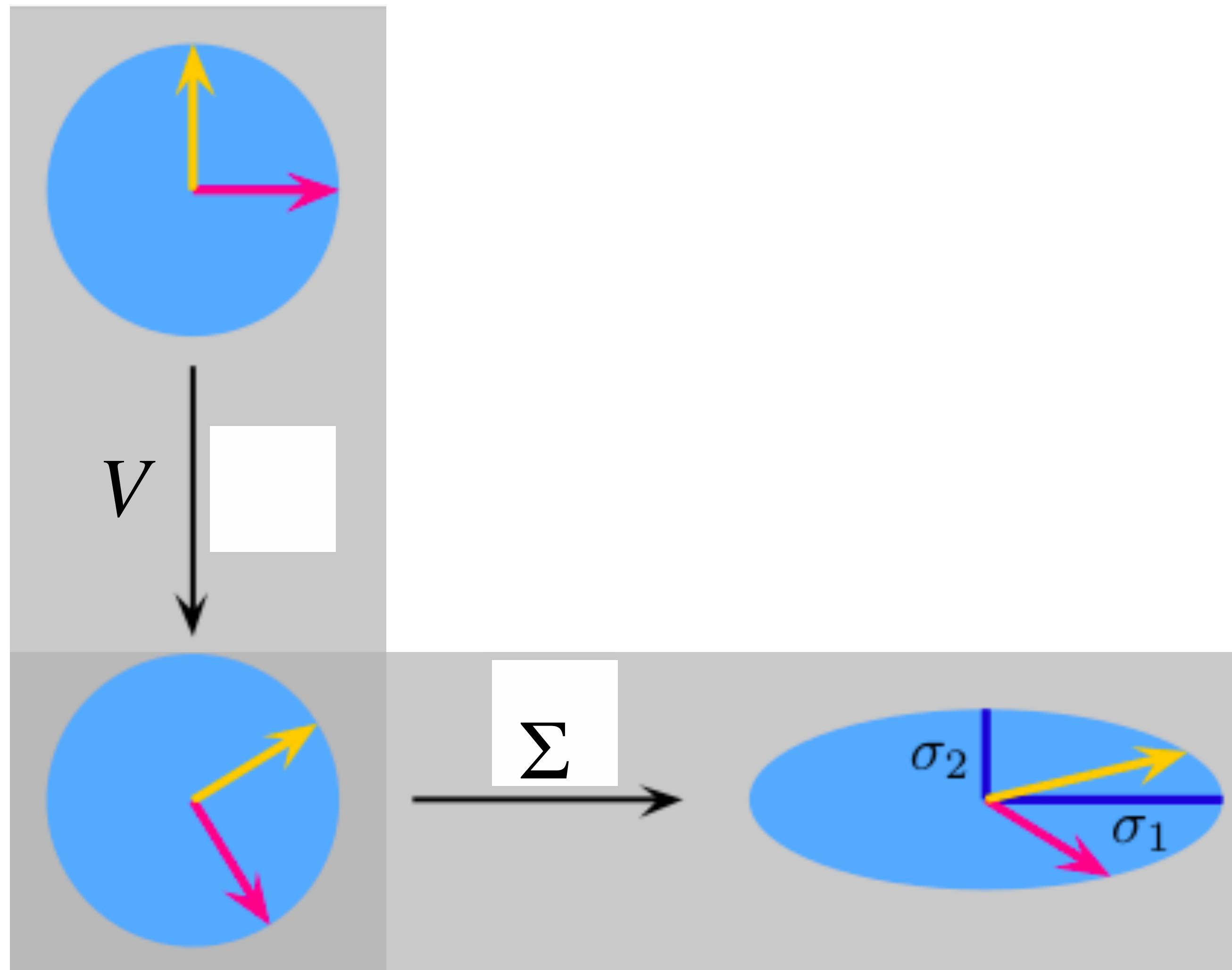
# Геометрический смысл



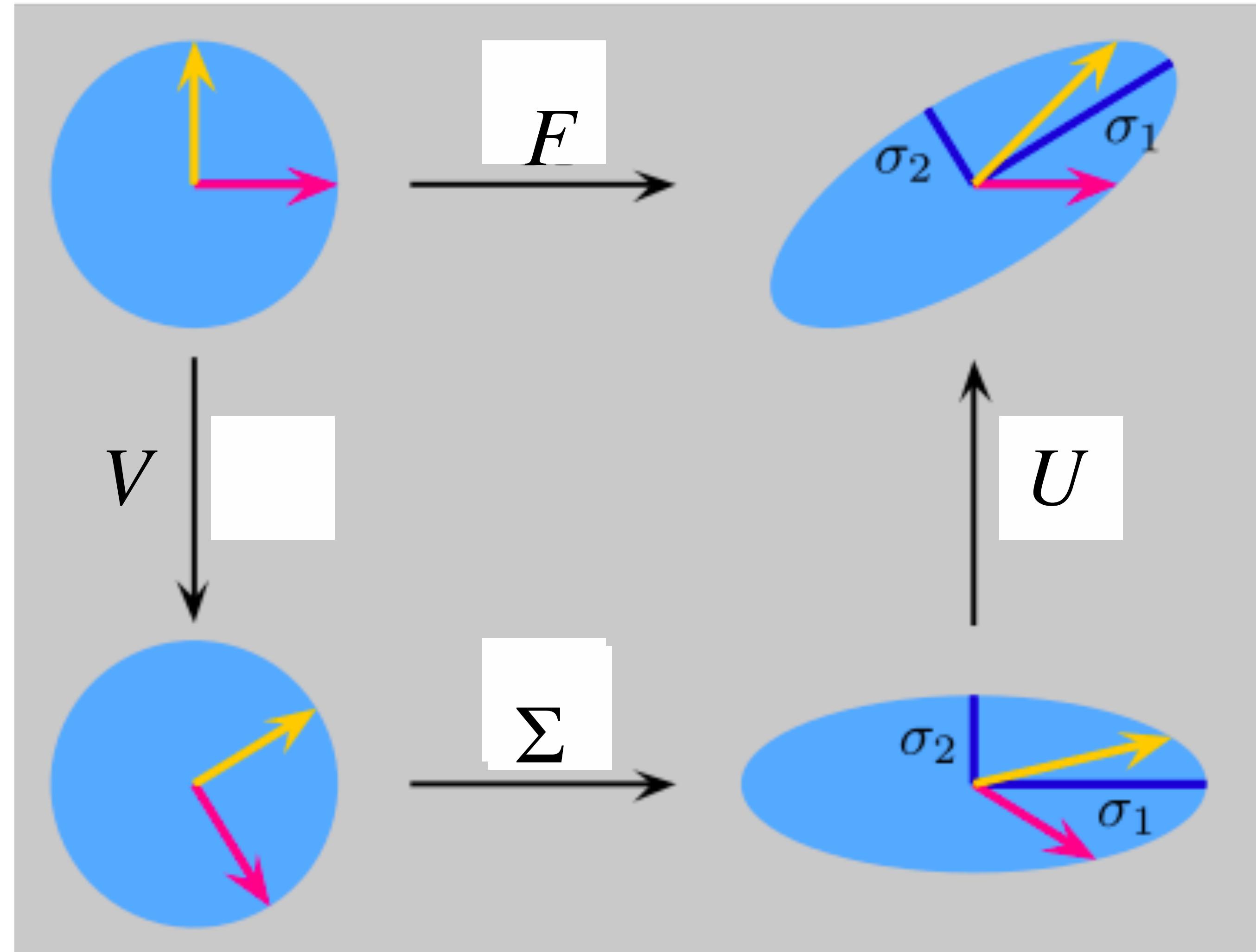
# Геометрический смысл



# Геометрический смысл



# Геометрический смысл



# Truncated SVD

Хотим приблизить матрицу  $F$  матрицей меньшего ранга

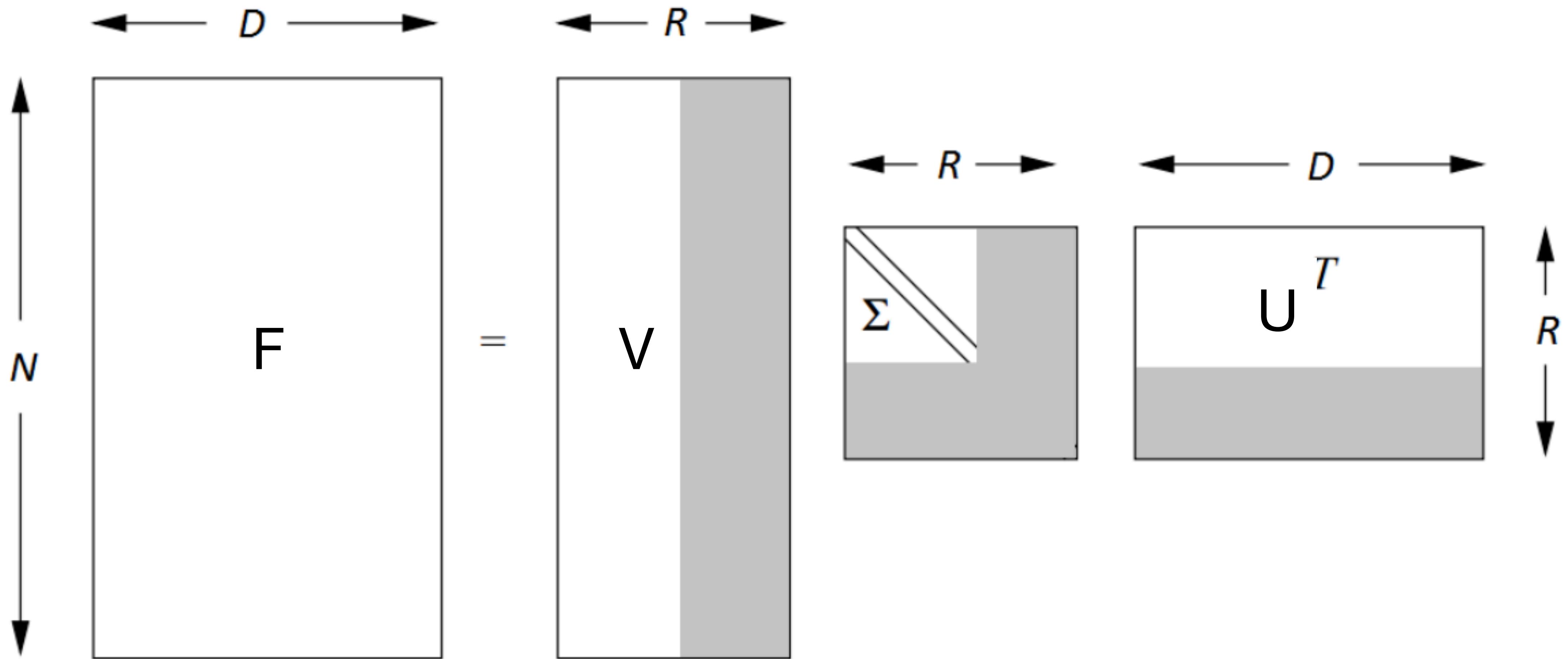
# Truncated SVD

Хотим приблизить матрицу  $F$  матрицей меньшего ранга

Можно взять  $t$  строк матрицы  $U$ ,  $t$  столбцов матрицы  $V$ .

С помощью такого подхода можно получить  
приближенное разложение матрицы  $F$

# Truncated SVD



# Principal component analysis



# Principal Component Analysis

Пусть есть матрица  $F$  объект-признак ( $l \times n$ ),  
где  $l$  - количество объектов,  $n$  - количество признаков

# Principal Component Analysis

Пусть есть матрица  $F$  объект-признак ( $l \times n$ ),  
где  $l$  - количество объектов,  $n$  - количество признаков

Хотим сократить количество признаков с  $n$  до  $m$

# Principal Component Analysis

Пусть есть матрица  $F$  объект-признак ( $l \times n$ ),  
где  $l$  - количество объектов,  $n$  - количество признаков

Хотим сократить количество признаков с  $n$  до  $m$

При этом старые признаки должны **линейно**  
восстанавливаться по новым, **как можно точнее** на  
обучающей выборке

# Principal Component Analysis

Старые признаки - линейная комбинация новых

$$\hat{f}_j(x) = \sum_{s=1}^m g_s(x) u_{js}, \quad j = 1, \dots, n, \quad \forall x \in X$$

# Principal Component Analysis

Старые признаки - линейная комбинация новых

$$\hat{f}_j(x) = \sum_{s=1}^m g_s(x) u_{js}, \quad j = 1, \dots, n, \quad \forall x \in X$$

Хотим восстанавливать старые признаки из новых, **как можно точнее**

$$\sum_{i=1}^{\ell} \sum_{j=1}^n (\hat{f}_j(x_i) - f_j(x_i))^2 \rightarrow \min_{\{g_s(x_i)\}, \{u_{js}\}}$$

# Principal Component Analysis

Матрицы «объекты–признаки», старая и новая:

$$F_{\ell \times \underline{n}} = \begin{pmatrix} f_1(x_1) & \dots & f_n(x_1) \\ \dots & \dots & \dots \\ f_1(x_\ell) & \dots & f_n(x_\ell) \end{pmatrix}; \quad G_{\ell \times \underline{m}} = \begin{pmatrix} g_1(x_1) & \dots & g_m(x_1) \\ \dots & \dots & \dots \\ g_1(x_\ell) & \dots & g_m(x_\ell) \end{pmatrix}$$

Матрица линейного преобразования новых признаков в старые:

$$U_{n \times m} = \begin{pmatrix} u_{11} & \dots & u_{1m} \\ \dots & \dots & \dots \\ u_{n1} & \dots & u_{nm} \end{pmatrix}$$

# Principal Component Analysis

$$\hat{F} = GU^\top \approx F$$

$$\sum_{i=1}^{\ell} \sum_{j=1}^n (\hat{f}_j(x_i) - f_j(x_i))^2 = \|GU^\top - F\|^2 \rightarrow \min_{\underline{G}, \underline{U}}$$

# Principal Component Analysis

Теорема:

Если  $m \leq \text{rk } F$ , то минимум  $\|GU^\top - F\|^2$  достигается, когда столбцы  $U$  — это с.в. матрицы  $F^\top F$ , соответствующие  $m$  максимальным с.з.  $\lambda_1, \dots, \lambda_m$ , а матрица  $G = FU$ .

# Principal Component Analysis

Теорема:

Если  $m \leq \text{rk } F$ , то минимум  $\|GU^\top - F\|^2$  достигается, когда столбцы  $U$  — это с.в. матрицы  $F^\top F$ , соответствующие  $m$  максимальным с.з.  $\lambda_1, \dots, \lambda_m$ , а матрица  $G = FU$ .

при этом:

матрица  $U$  ортонормирована:  $U^\top U = I_m$ ;

матрица  $G$  ортогональна:  $G^\top G = \Lambda = \text{diag}(\lambda_1, \dots, \lambda_m)$

$$U\Lambda = F^\top FU; \quad G\Lambda = FF^\top G;$$

$$\|GU^\top - F\|^2 = \|F\|^2 - \text{tr } \Lambda = \sum_{j=m+1}^n \lambda_j.$$

# PCA vs. SVD

Чем отличается PCA от SVD ?

**Ничем, если  $m = n$**

# PCA vs. SVD

**SVD:**

Если взять  $m = n$ , то:

$$\|GU^\top - F\|^2 = 0;$$

представление  $\hat{F} = GU^\top = F$  точное и совпадает с сингулярным разложением при  $G = V\sqrt{\Lambda}$ :

$$F = GU^\top = V\sqrt{\Lambda}U^\top; \quad U^\top U = I_m; \quad V^\top V = I_m.$$

$$F = V\Sigma U^T$$

$$\Sigma = \sqrt{\Lambda}$$

# PCA vs. SVD

**SVD:**

Если взять  $m = n$ , то:

$$\|GU^\top - F\|^2 = 0;$$

представление  $\hat{F} = GU^\top = F$  точное и совпадает с сингулярным разложением при  $G = V\sqrt{\Lambda}$ :

$$F = GU^\top = V\sqrt{\Lambda}U^\top; \quad U^\top U = I_m; \quad V^\top V = I_m.$$

$$F = V\Sigma U^T$$

$$\Sigma = \sqrt{\Lambda}$$

Поскольку новые признаки некоррелированы ( $G^\top G = \Lambda$ ), преобразование  $U$  называется *декоррелирующим*

# Как выбрать $m$

Упорядочиваем собственные числа по убыванию:  $\lambda_1 \geq \dots \geq \lambda_n \geq 0$ .

# Как выбрать $m$

Упорядочиваем собственные числа по убыванию:  $\lambda_1 \geq \dots \geq \lambda_n \geq 0$ .

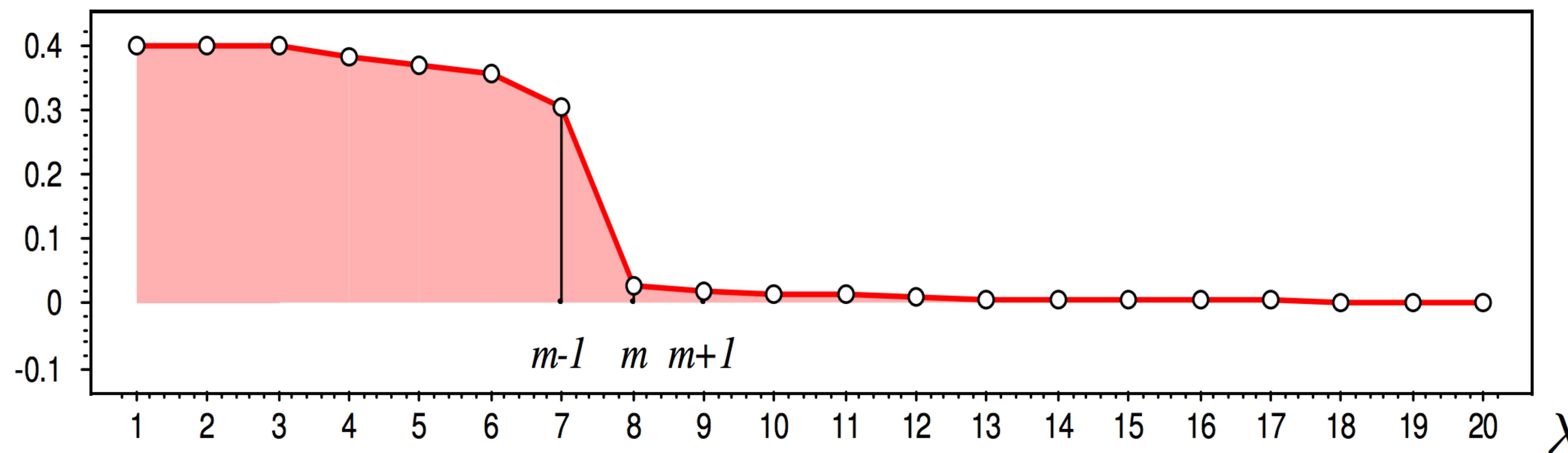
$$E_m = \frac{\|GU^\top - F\|^2}{\|F\|^2} = \frac{\lambda_{m+1} + \dots + \lambda_n}{\lambda_1 + \dots + \lambda_n} \leq \varepsilon.$$

# Как выбрать $m$

Упорядочиваем собственные числа по убыванию:  $\lambda_1 \geq \dots \geq \lambda_n \geq 0$ .

$$E_m = \frac{\|GU^\top - F\|^2}{\|F\|^2} = \frac{\lambda_{m+1} + \dots + \lambda_n}{\lambda_1 + \dots + \lambda_n} \leq \varepsilon.$$

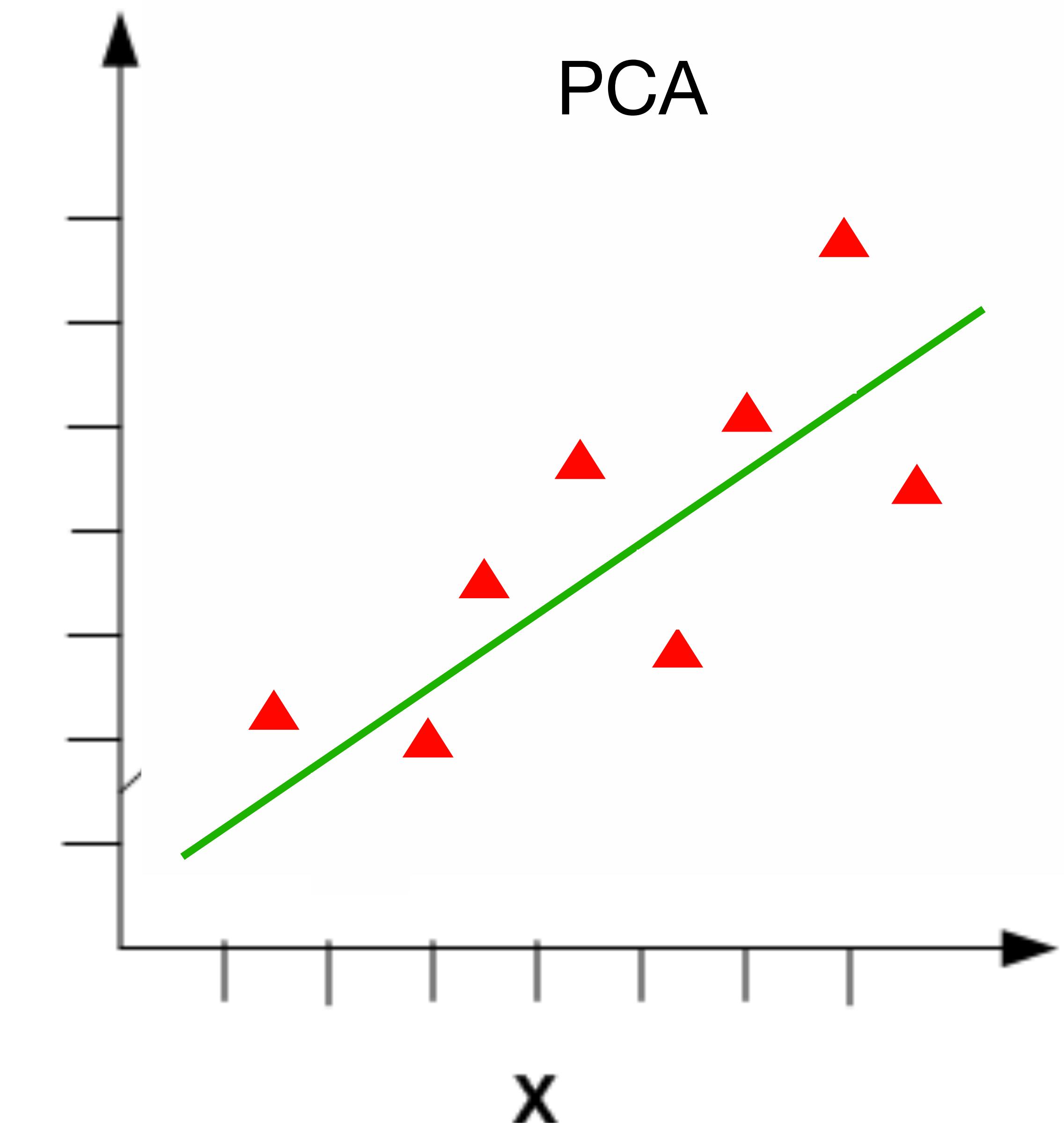
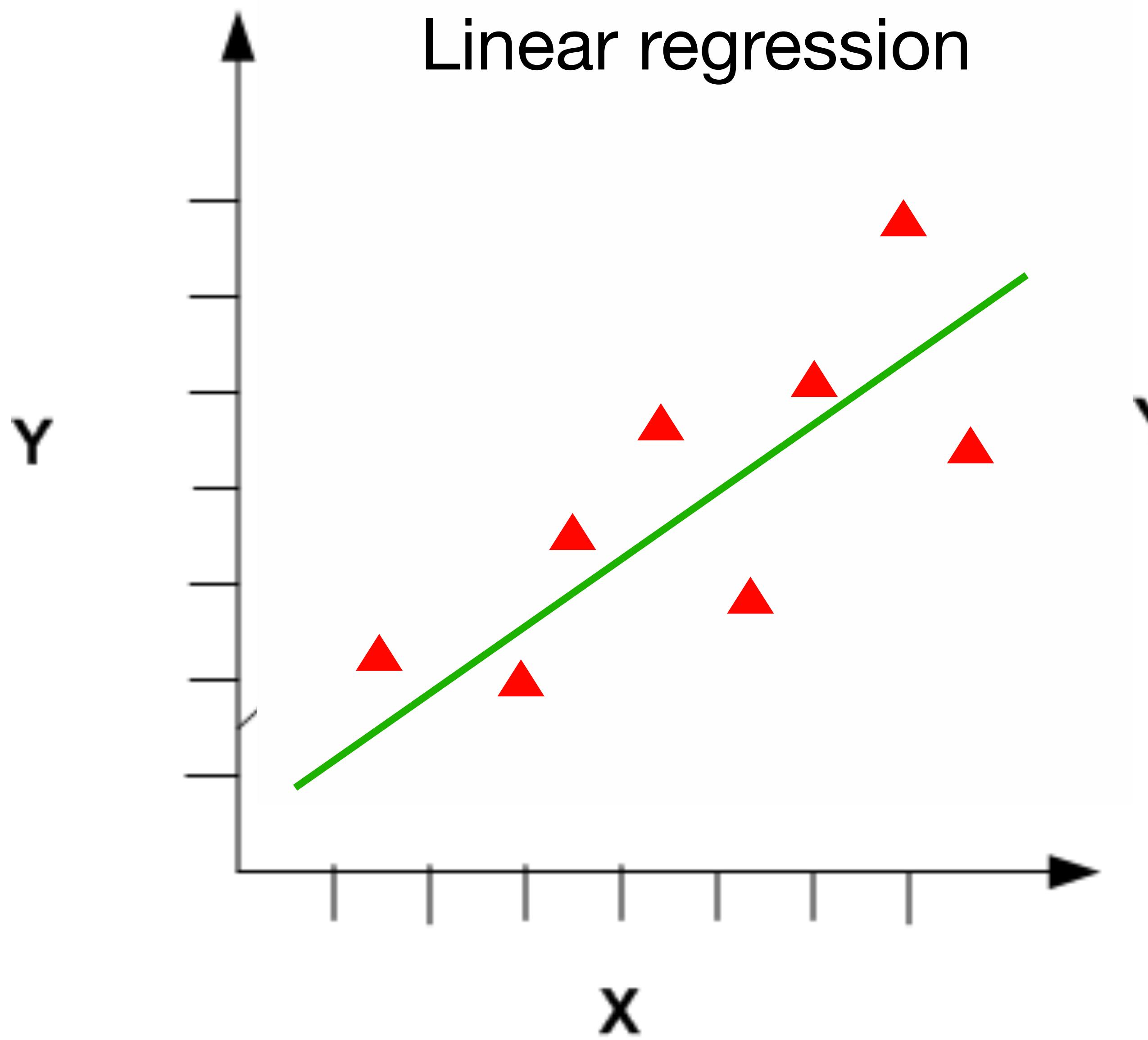
*Критерий «крутого склона»:* находим  $m$ :  $E_{m-1} \gg E_m$ :



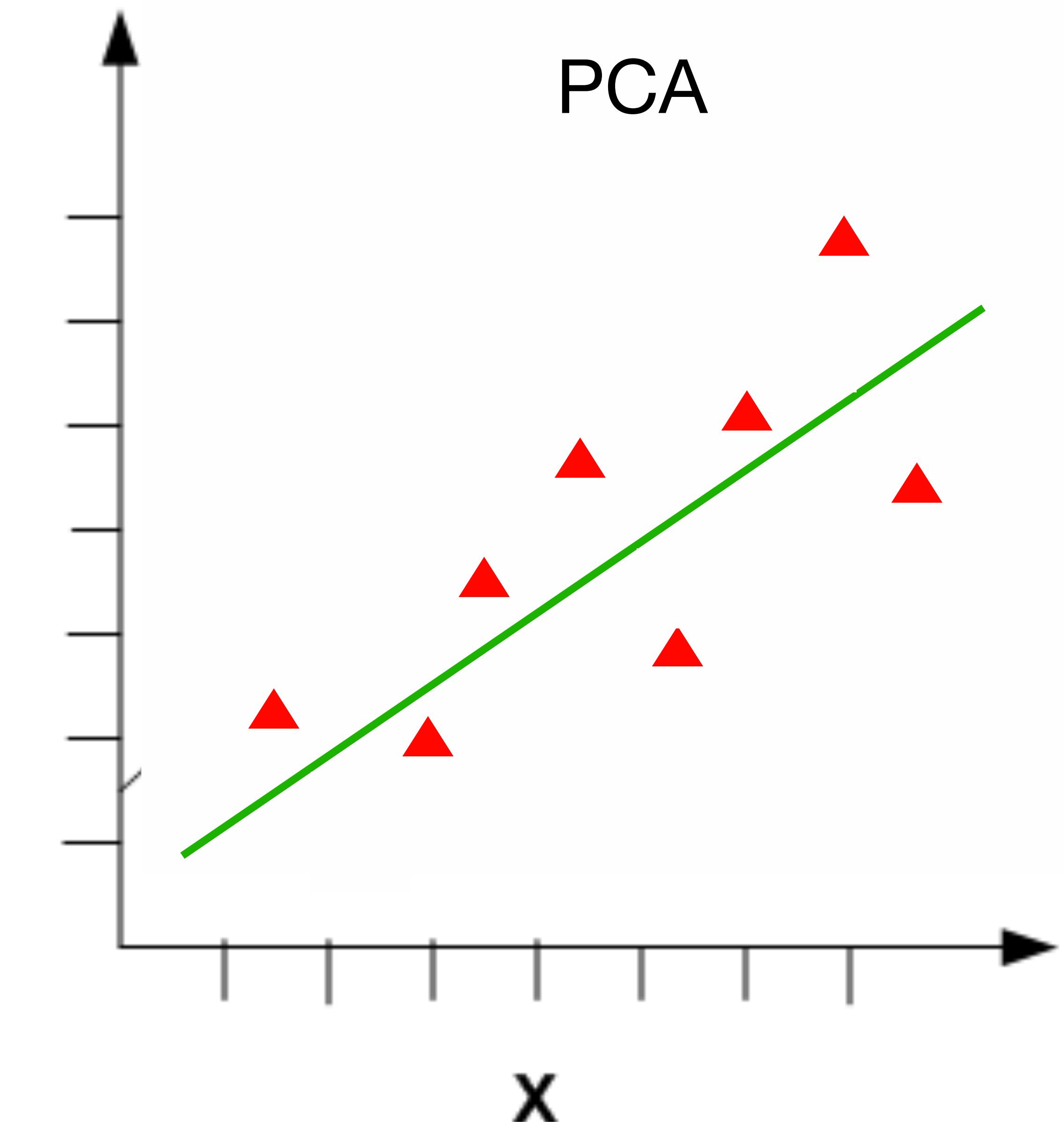
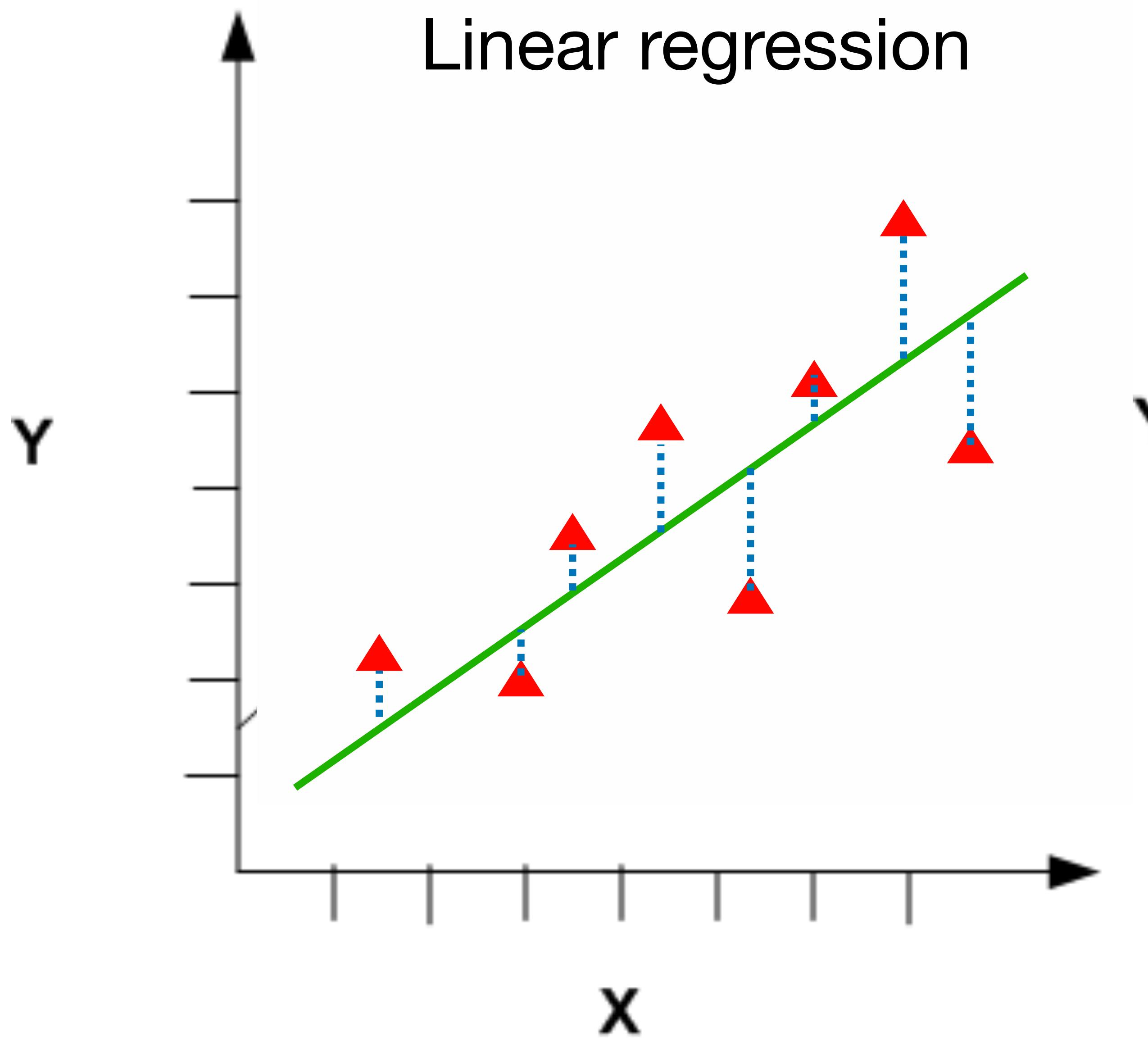
# PCA & SVD



# More intuitions

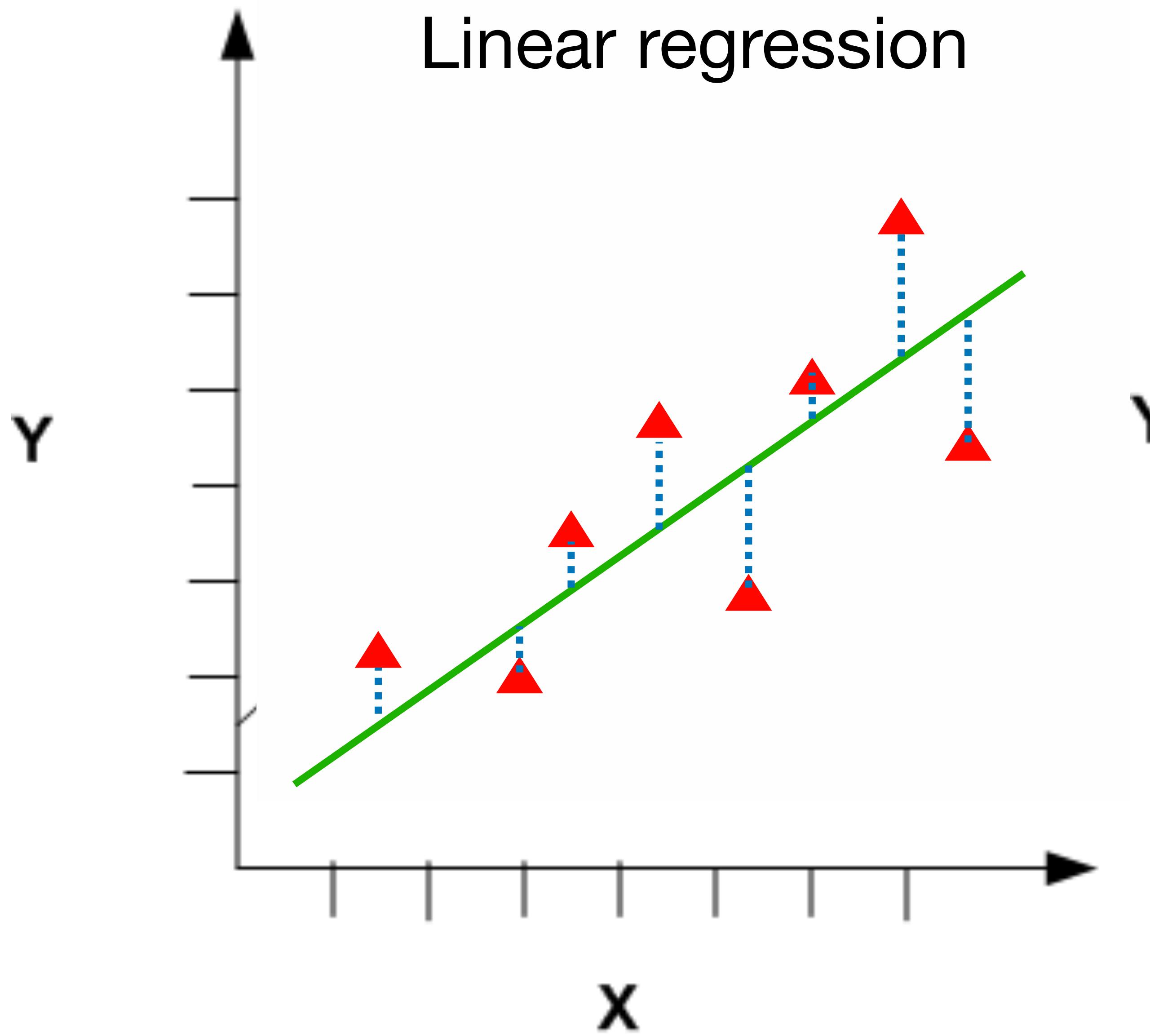


# More intuitions

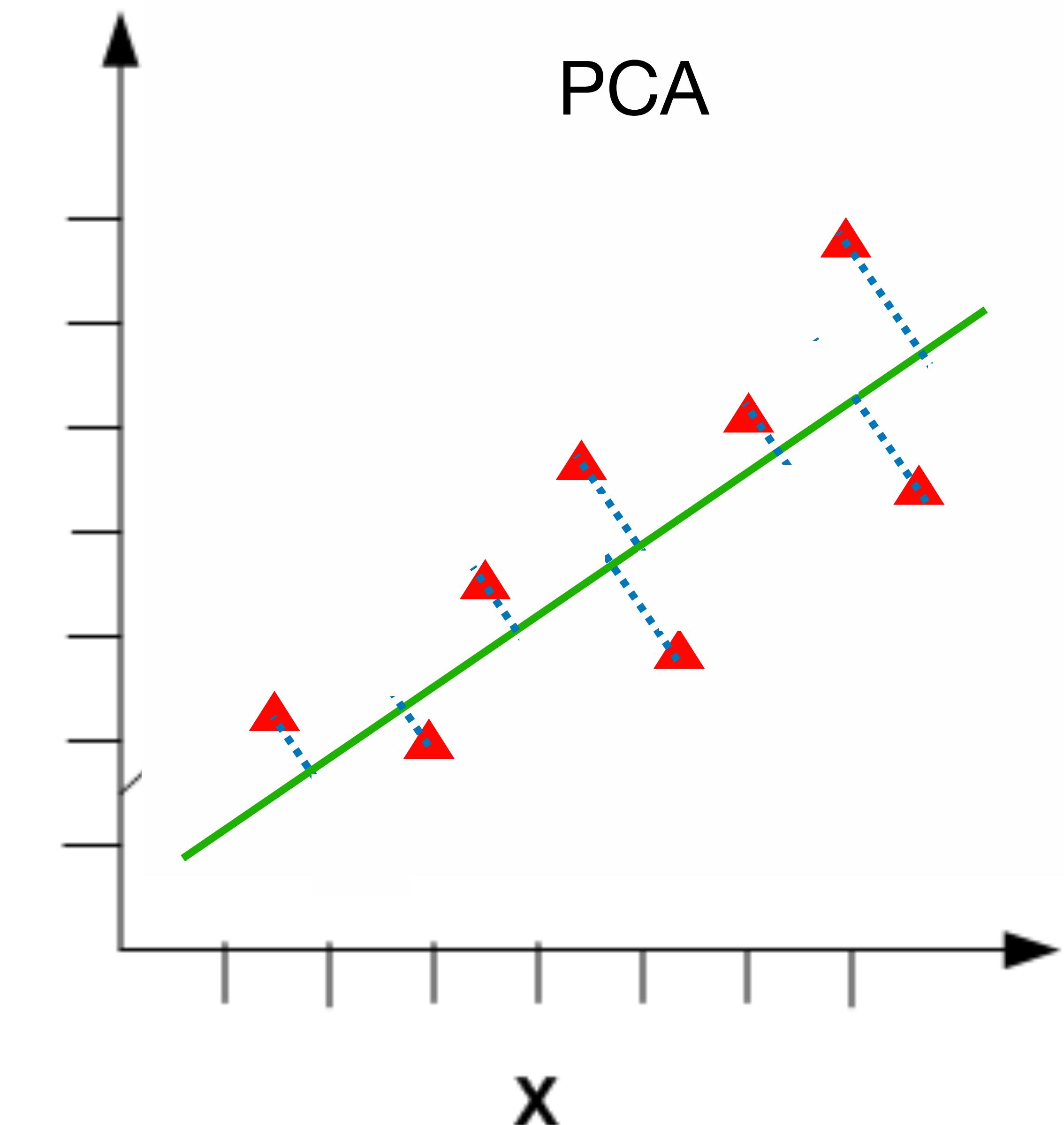


# More intuitions

Linear regression



PCA



# More intuitions

- РСА ищет такие отображения (прямые, плоскости),  
**суммарное расстояние до которых от точек выборки  
будет минимально**

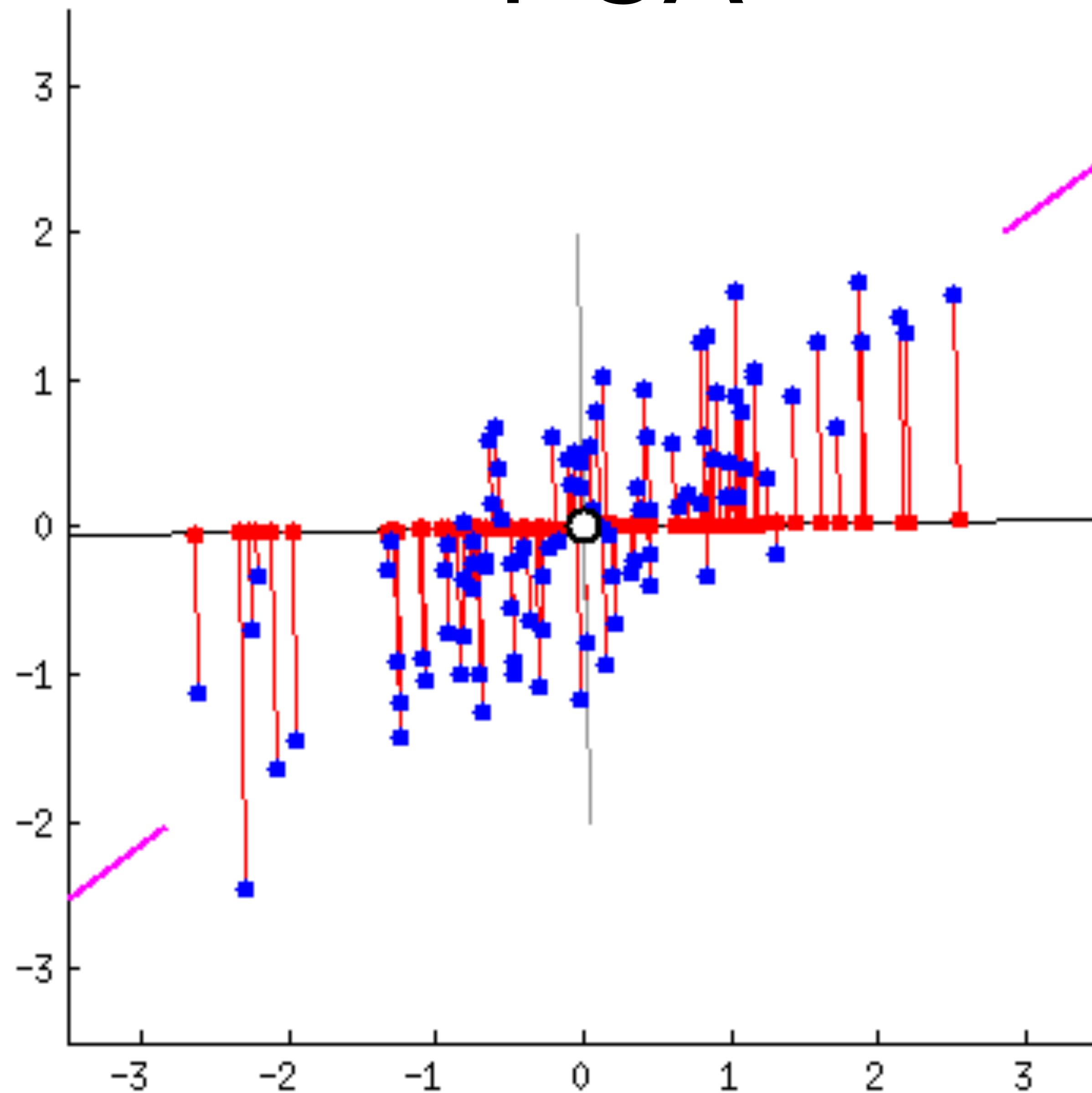
# More intuitions

- РСА ищет такие отображения (прямые, плоскости), **суммарное расстояние до которых от точек выборки будет минимально**
- РСА ищет такие ортогональные проекции, **дисперсия вдоль которых для точек выборки будет максимальна**

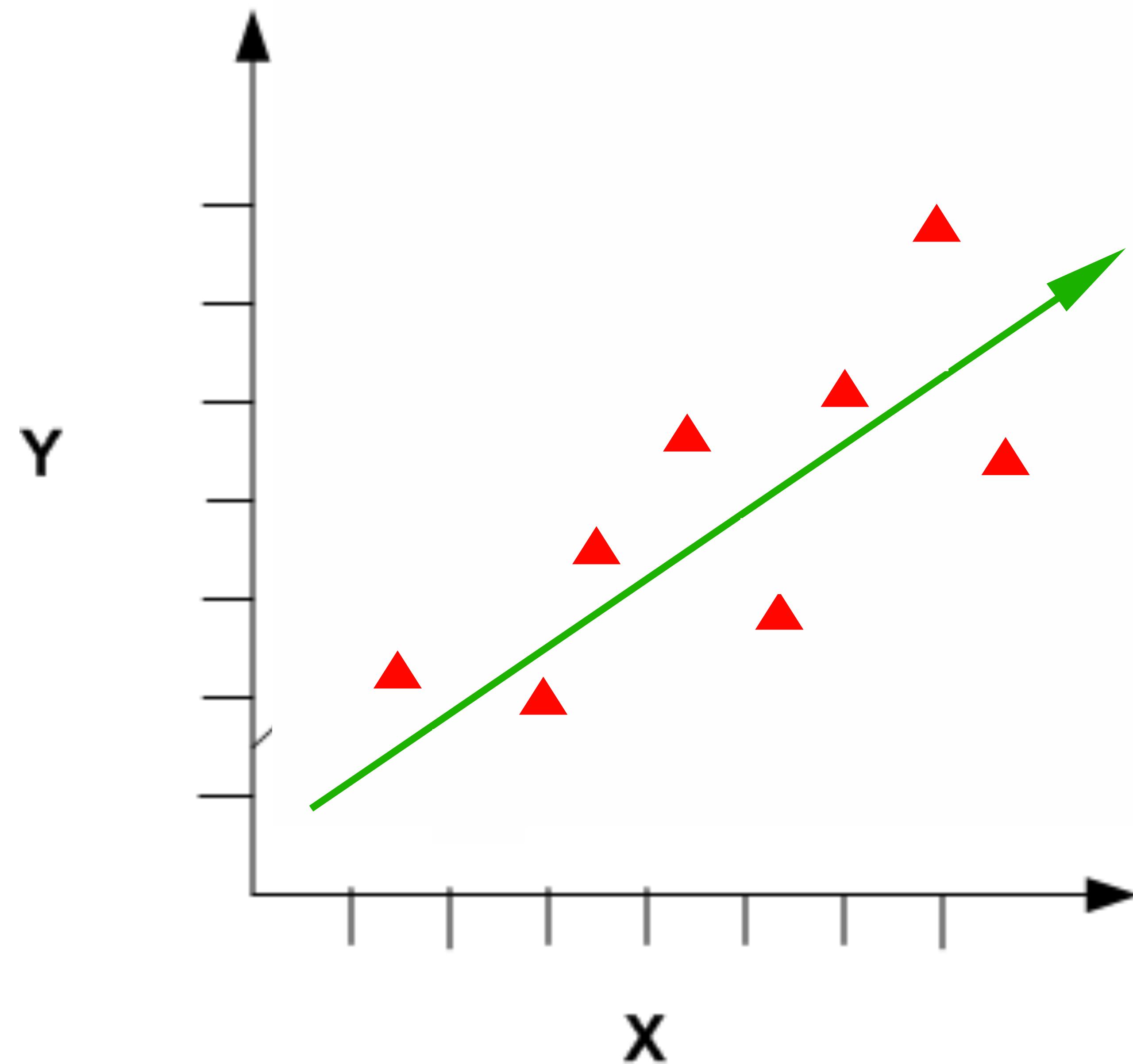
# More intuitions

- РСА ищет такие отображения (прямые, плоскости), **суммарное расстояние до которых от точек выборки будет минимально**
- РСА ищет такие ортогональные проекции, **дисперсия вдоль которых для точек выборки будет максимальна**
- РСА строит такой базис, в котором **новые признаки ортогональны**

# PCA

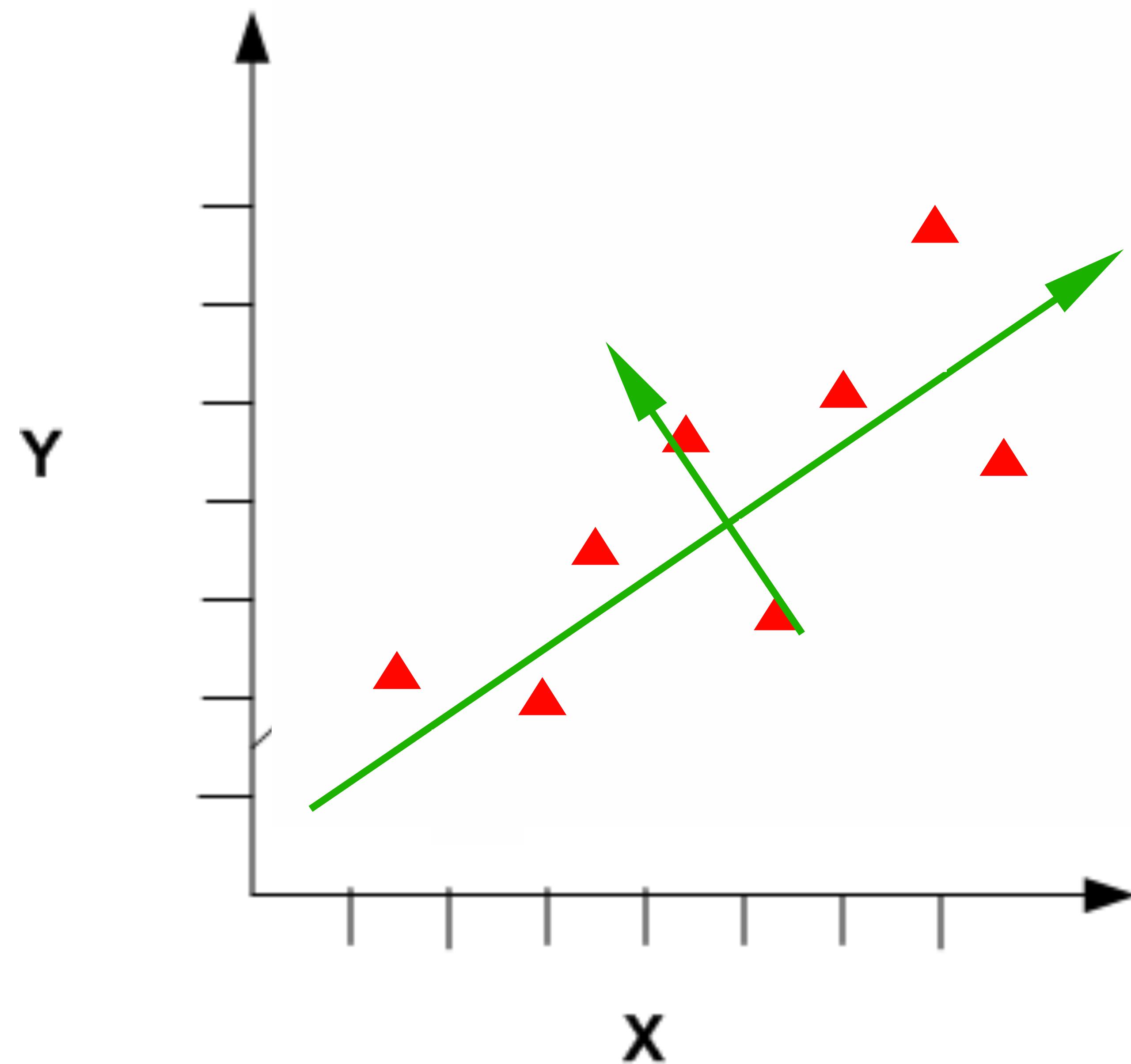


# PCA



Где вторая  
главная  
компоненты?

# PCA



Где вторая  
главная  
компоненты?

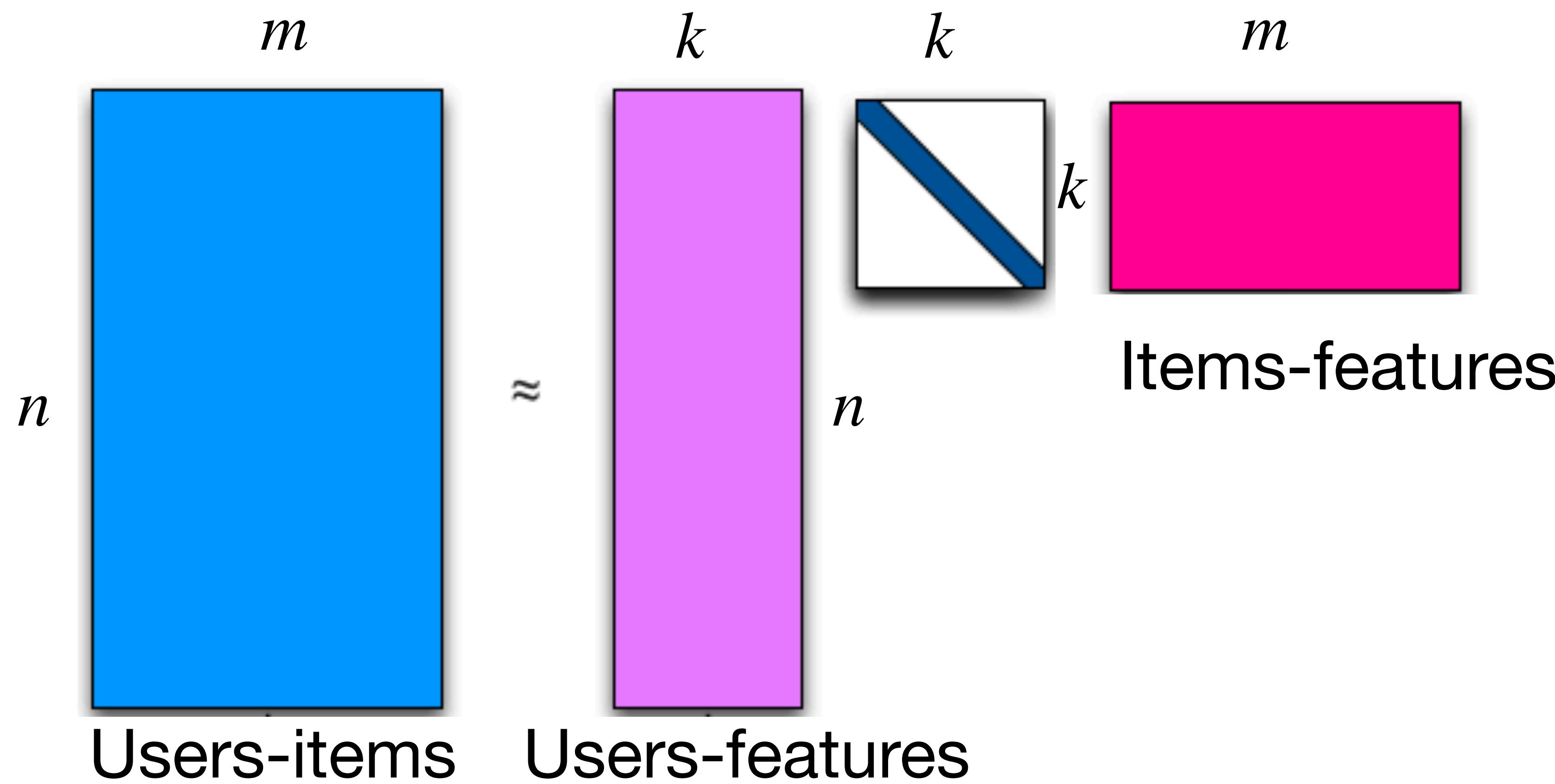
# SVD & PCA applications

Применяются не только для снижения размерности

# SVD & PCA applications

Применяются не только для снижения размерности

Но и, например, для рекомендаций:

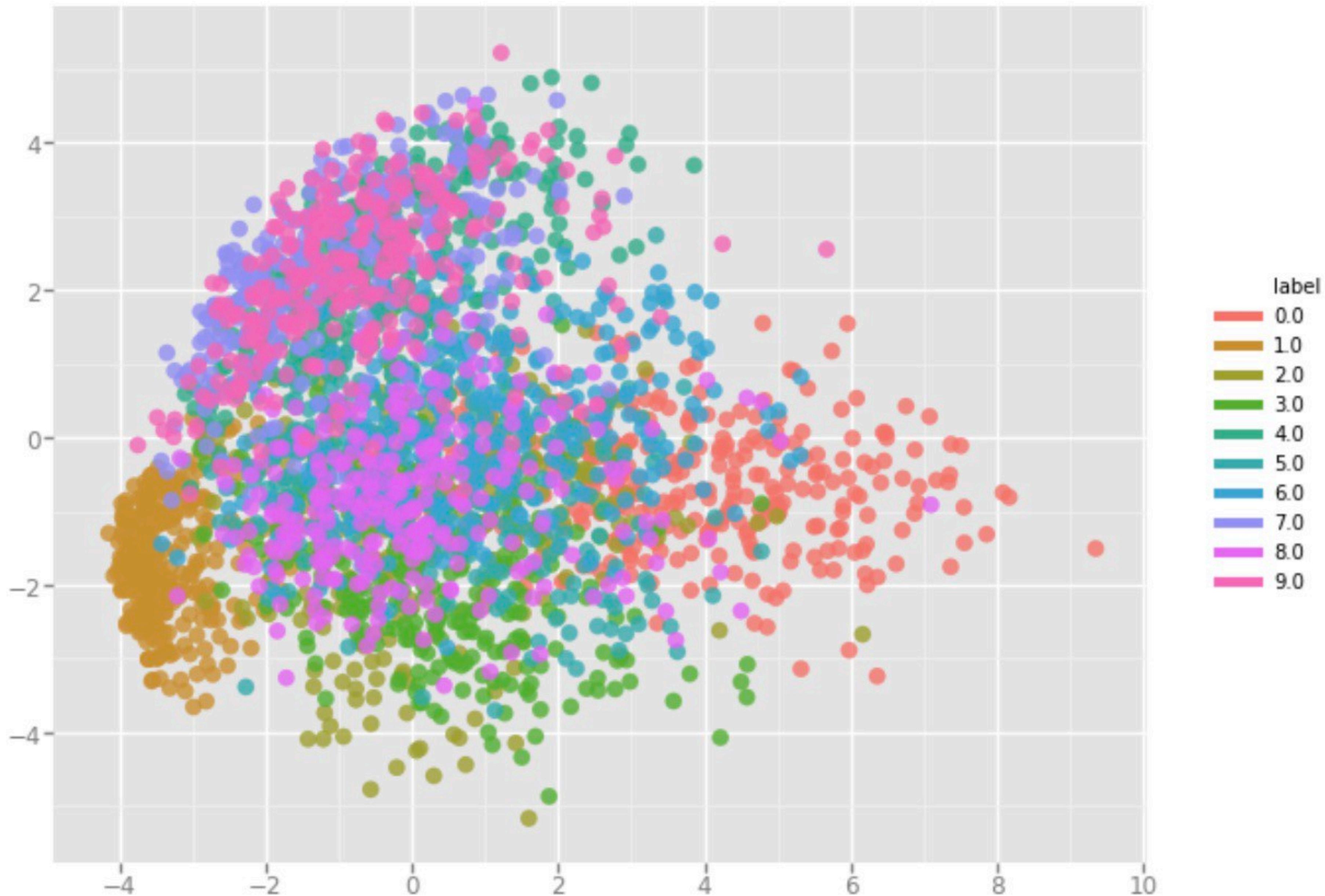


# SVD & PCA applications

А также для визуализации

(хотя сейчас чаще применяют нелинейные  
методы типа t-SNE)

# SVD & PCA applications



# Random Projections

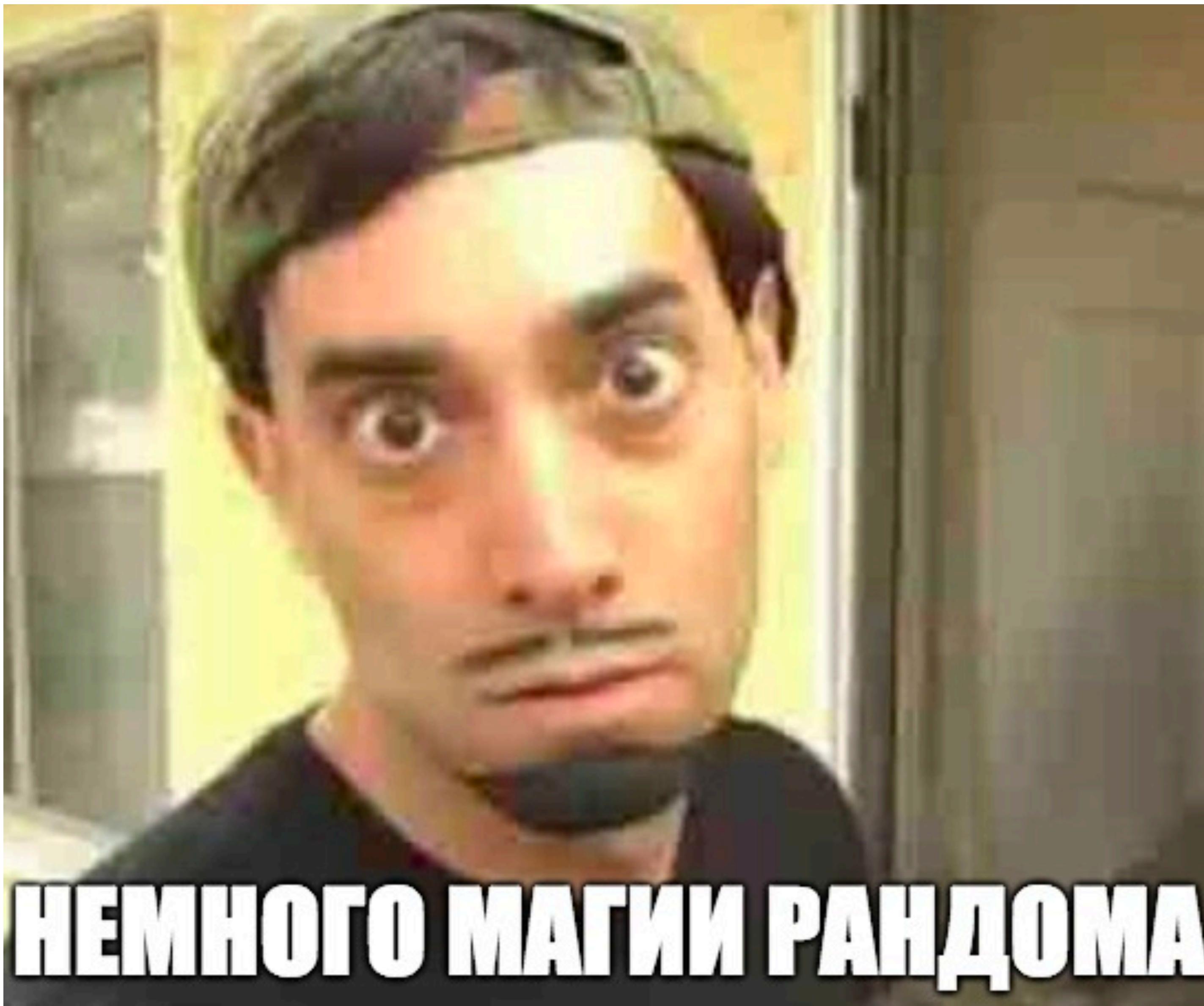


# Random projections

Если в выборке **мало объектов и много признаков**, то ее можно спроектировать в пространство меньшей размерности так, что **расстояния** между объектами **слабо изменяются**

Метод случайных проекций - простейший линейный подход

# Random projections



# Random projections

Формируем и фиксируем матрицу  $W$  из случайных малых чисел

$$w_{ij} \sim \mathcal{N}(0, \frac{1}{d})$$

# Random projections

Формируем и фиксируем матрицу  $W$  из случайных малых чисел

$$w_{ij} \sim \mathcal{N}(0, \frac{1}{d})$$

Получаем новые признаки с помощью матричного умножения

$$z_{ij} = \sum_{k=1}^D w_{jk} x_{ik}$$

# Manifold Learning



# Manifold Learning

PCA - линейный метод снижения размерности признакового пространства

Но никто нас не ограничивает в выборе преобразования

Есть ряд нелинейных методов снижения размерности:  
Isomap, MDS, t-SNE

# MDS - multidimensional scaling

Идея: при снижении размерности **сохранять попарные расстояния** между объектами

# MDS - multidimensional scaling

Идея: при снижении размерности **сохранять попарные расстояния** между объектами

$x_1, \dots, x_\ell$  - исходные объекты

$\tilde{x}_1, \dots, \tilde{x}_\ell$  - объекты в новом признаковом пространстве

# MDS - multidimensional scaling

Идея: при снижении размерности **сохранять попарные расстояния** между объектами

$x_1, \dots, x_\ell$  - исходные объекты

$\tilde{x}_1, \dots, \tilde{x}_\ell$  - объекты в новом признаковом пространстве

$$\sum_{i < j}^{\ell} (\tilde{d}_{ij} - d_{ij})^2 \rightarrow \min_{\tilde{x}_1, \dots, \tilde{x}_\ell}$$

# t-SNE

При решении задачи MDS очень **сложно сохранить** попарные расстояния в случае **сильного снижения** размерности

# t-SNE

При решении задачи MDS очень **сложно сохранить** попарные расстояния в случае **сильного снижения** размерности

Идея: при снижении размерности **сохранять пропорции расстояний** между объектами

# t-SNE

При решении задачи MDS очень **сложно сохранить** попарные расстояния в случае **сильного снижения** размерности

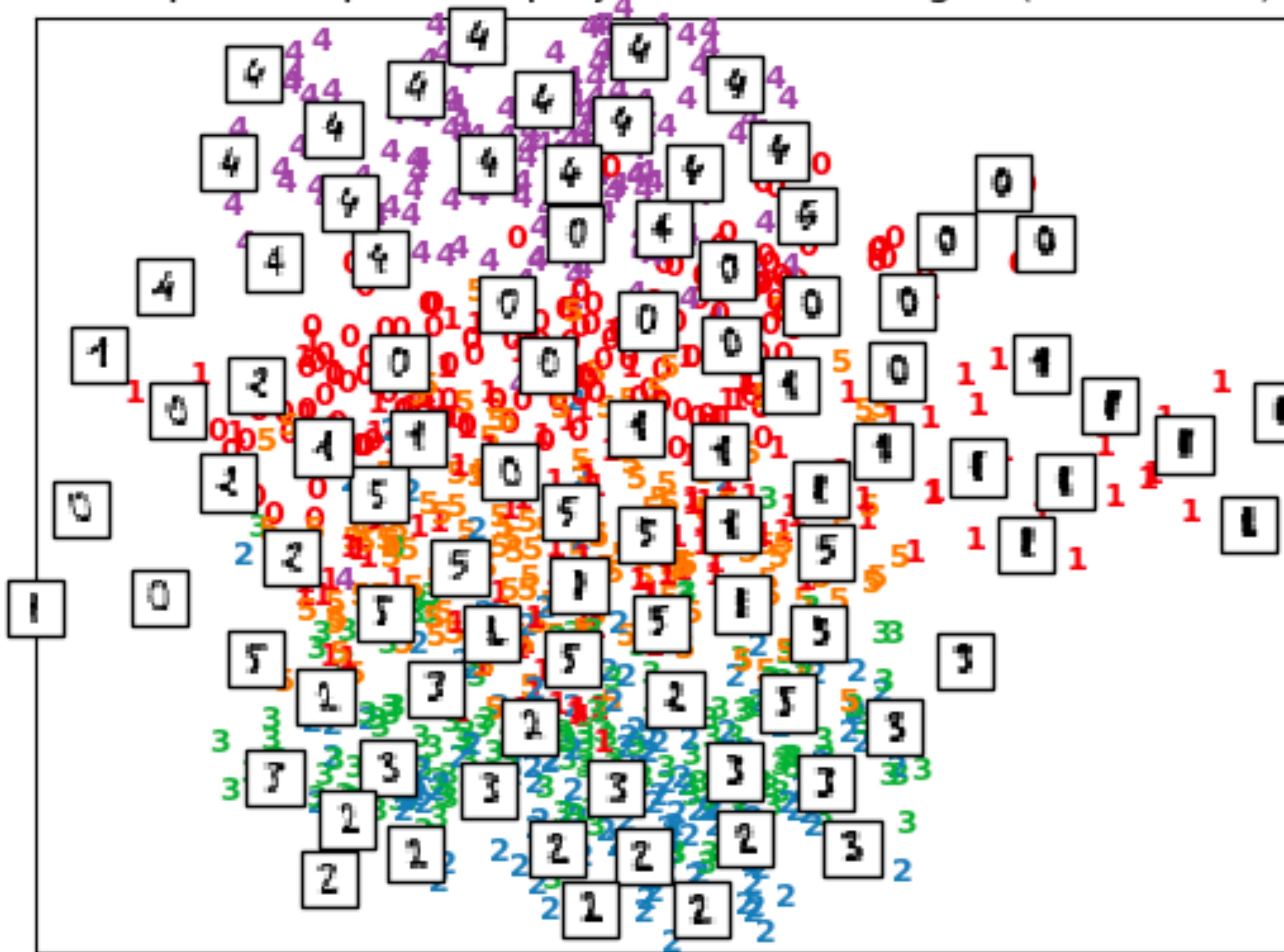
Идея: при снижении размерности **сохранять пропорции расстояний** между объектами

$\rho(x_i, x_j) = \alpha \rho(x_i, x_k)$  - в исходном пространстве признаков

$\rho(\tilde{x}_i, \tilde{x}_j) = \alpha \rho(\tilde{x}_i, \tilde{x}_k)$  - в новом пространстве признаков

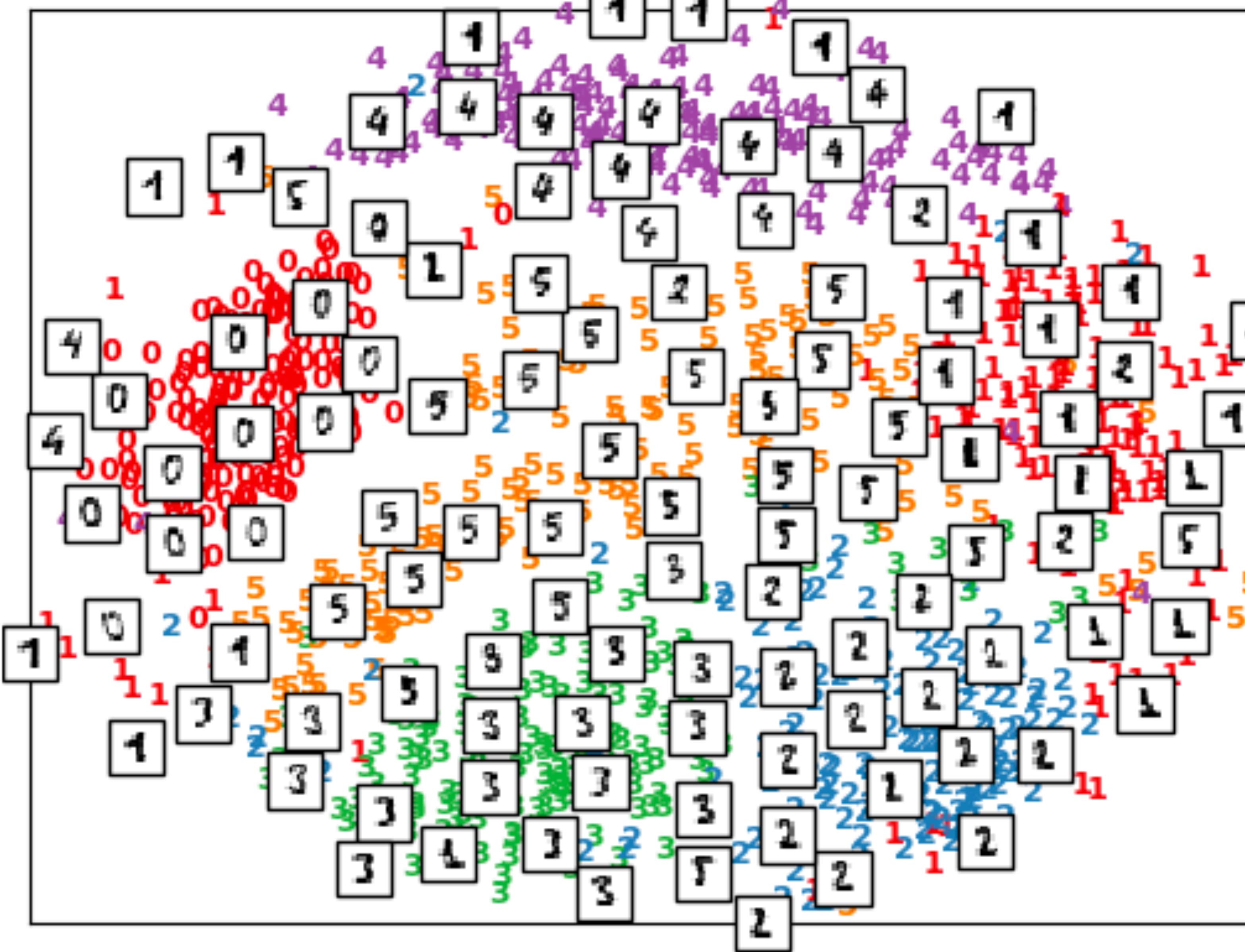
# MNIST Visualisation

Principal Components projection of the digits (time 0.00s)



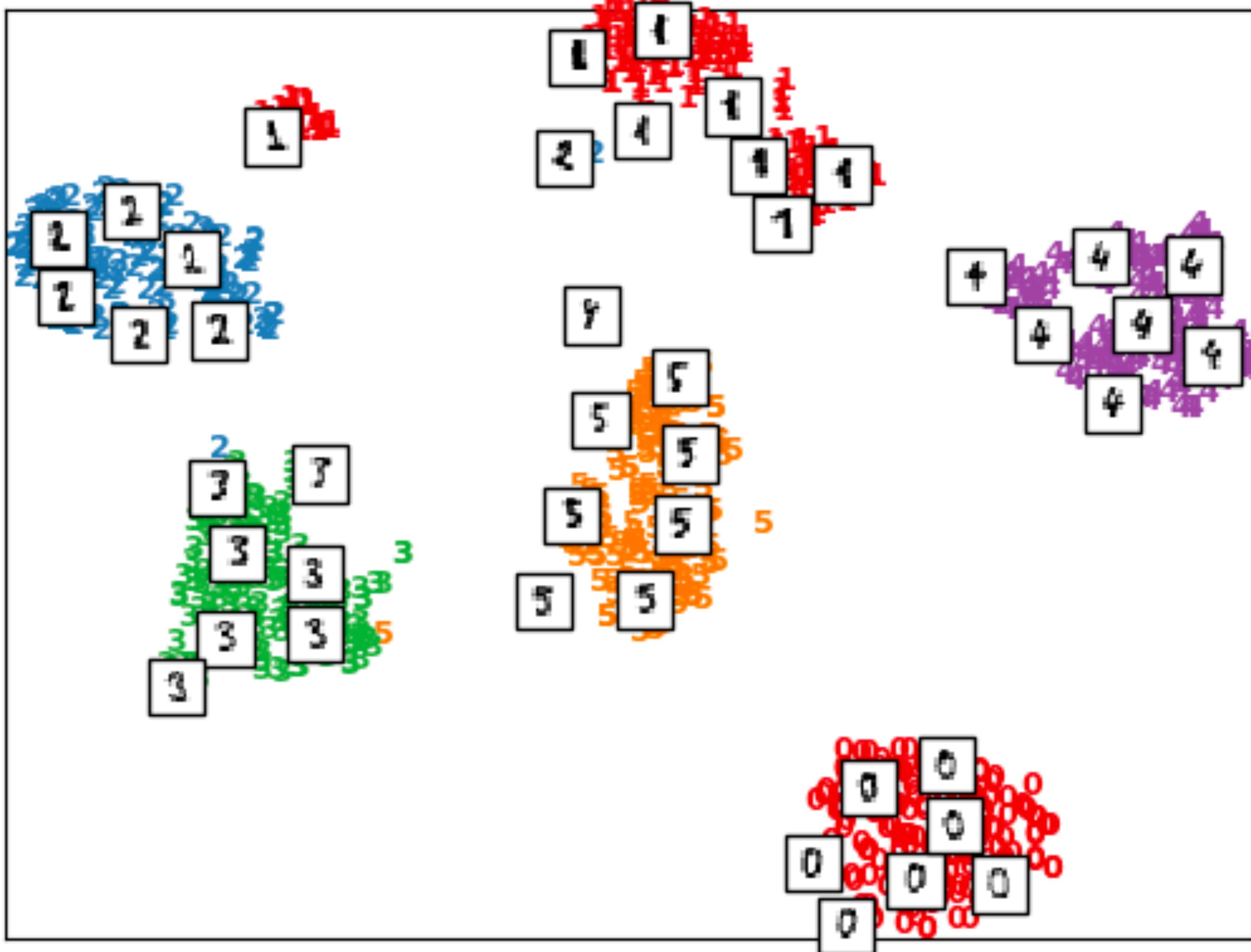
# MNIST Visualisation

MDS embedding of the digits (time 3.49s)



# MNIST Visualisation

t-SNE embedding of the digits (time 5.12s)



# Признаковое описание текста и снижение размерности

# Признаковое описание текста

Как человек определяет, что текст - про футбол?

# Признаковое описание текста

Как человек определяет, что текст - про футбол?

**Упрощенно:** текст содержит термины про футбол



# Признаковое описание текста

Самый простой подход - бинарные признаки наличия слова в документе - **one-hot-encoding**

# Признаковое описание текста

Самый простой подход - бинарные признаки наличия слова в документе - **one-hot-encoding**

*Предположим, что мы обрабатываем коллекцию документов.*

**Как будет выглядеть представление документа?**

# Признаковое описание текста

Самый простой подход - бинарные признаки наличия слова в документе - **one-hot-encoding**

*Предположим, что мы обрабатываем коллекцию документов.*

**Как будет выглядеть представление документа?**

Вектор с размерностью, равной количеству уникальных слов в коллекции

# Признаковое описание текста



# Признаковое описание текста

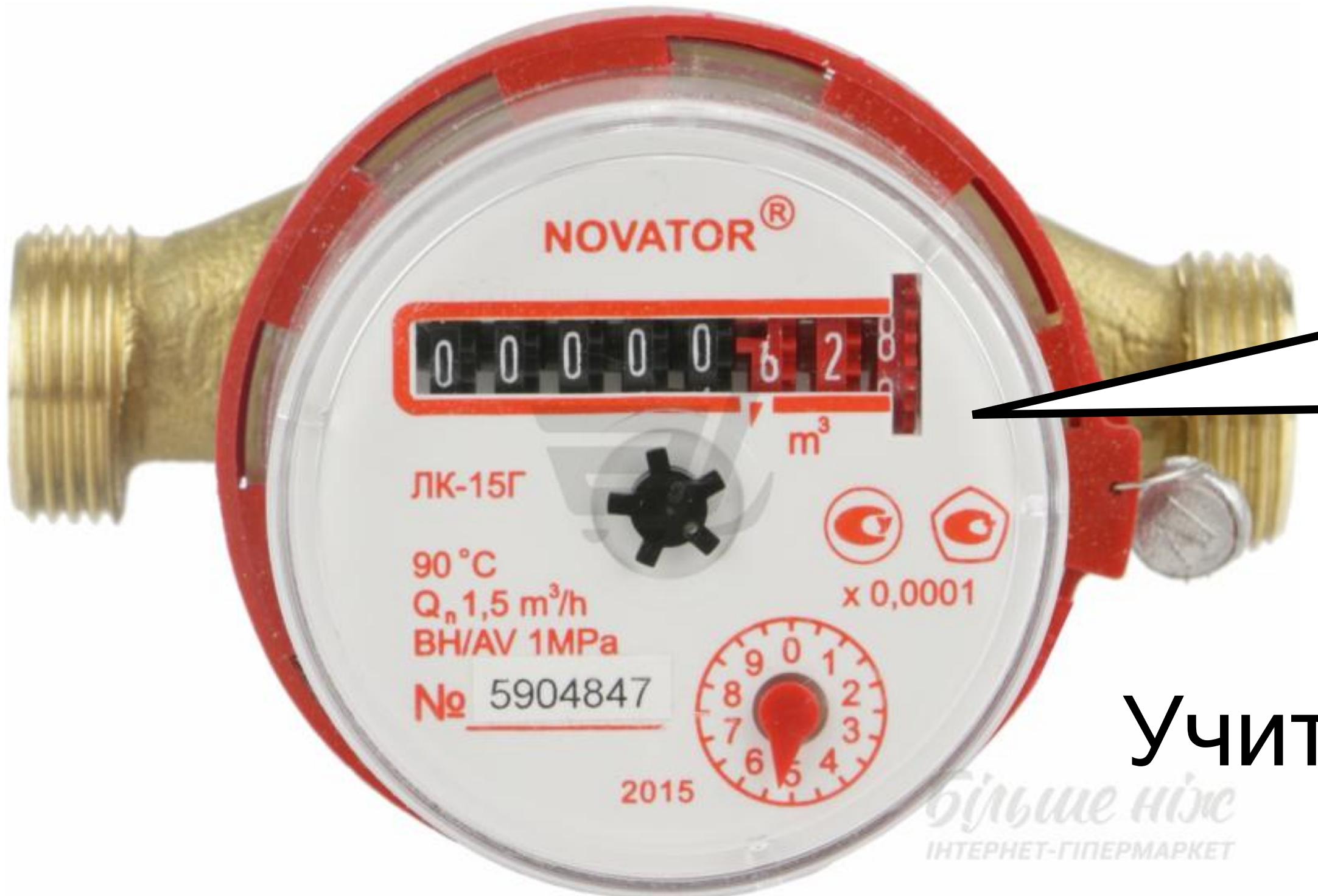
Логичнее, что чем **чаще** слово встречается, тем оно **важнее**

Давайте хранить не бинарный признак, а **счетчик**.

# Признаковое описание текста

Логичнее, что чем **чаще** слово встречается, тем оно **важнее**

Давайте хранить не бинарный признак, а **счетчик**.



***scikit-learn:  
CountVectorizer***

Учитывает частоту терма

# Признаковое описание текста

## CountVectorizer

	о	улыбка	мама	я	боже	голова	кругом
о боже мама мама я схожу с ума	1	0	2	1	1	0	0
ее улыбка мама кругом голова	0	1	1	0	0	1	1

Каждому тексту соответствует **вектор**

# Признаковое описание текста

**Интуитивно:** слова, встречающие в **каждом** документе - **неинформационны**.

Наиболее всего полезны слова, встречающиеся довольно часто в **небольшом количестве** документов.

# TF - term frequency

weighting scheme	tf weight
binary	0, 1
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$

# TF - term frequency

weighting scheme	tf weight
binary	0, 1
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$

Зачем нормализуют ?

# DF - document frequency

**В скольки документах** корпуса встречается данный терм.

$|\{d_i \in D \mid t \in d_i\}|$  - число документов из коллекции D, в которых встречается t

# DF - document frequency

**В скольки документах** корпуса встречается данный терм.

$|\{d_i \in D | t \in d_i\}|$  - число документов из коллекции D, в которых встречается t

$$df = \frac{|\{d_i \in D | t \in d_i\}|}{|D|}$$

$$idf = \frac{1}{df} = \frac{|D|}{|\{d_i \in D | t \in d_i\}|}$$

$$idf = \log\left(\frac{1}{df}\right) = \log\left(\frac{|D|}{|\{d_i \in D | t \in d_i\}|}\right)$$

- сглаживание

# TF-IDF

$$TF\_IDF = tf \times idf$$



# TF-IDF

$$TF\_IDF = tf \times idf = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} * \log\left(\frac{|D|}{|\{d_i \in D | t \in d_i\}|}\right)$$

***scikit-learn: TfidfVectorizer***

Учитывается ли порядок слов в предложении?



# Bag of words

Учитывается ли порядок слов в предложении?

Нет, документ - «**мешок слов**»

# Bag of words

Учитывается ли порядок слов в предложении?

Нет, документ - «**мешок слов**»

- 1) «Я люблю **помидоры** и не люблю **читать**»
- 2) «Я люблю **читать** и не люблю **помидоры**»

Один и тот же вектор для обоих предложений



# Bag of words

Учитывается ли порядок слов в предложении?

Нет, документ - «**мешок слов**»

- 1) «Я люблю **помидоры** и не люблю **читать**»
- 2) «Я люблю **читать** и не люблю **помидоры**»

Один и тот же вектор для обоих предложений

Как можно исправить?



# Bag of words

Самый простой хак - добавить **словесные нграммы**

# Bag of words

Самый простой хак - добавить **словесные нграммы**

- 1) «Я люблю помидоры и не люблю читать»
- 2) «Я люблю читать и не люблю помидоры»

# Bag of words

Самый простой хак - добавить **словесные нграммы**

- 1) «Я люблю помидоры и не люблю читать»
- 2) «Я люблю читать и не люблю помидоры»

Чем плохо?



# Bag of words

Самый простой хак - добавить **словесные нграммы**

- 1) «Я люблю помидоры и не люблю читать»
- 2) «Я люблю читать и не люблю помидоры»

Чем плохо?

Размерность вектора текста увеличивается в разы

# Bag of words



зараза

# Bag of words

Можно ограничиться только частотными нграммами

Это работает нормально и не так сильно раздувает вектор

```
TfidfVectorizer(ngram_range=(1,3), min_df=2, max_df=0.5)
```

# Общие проблемы подхода ВОW

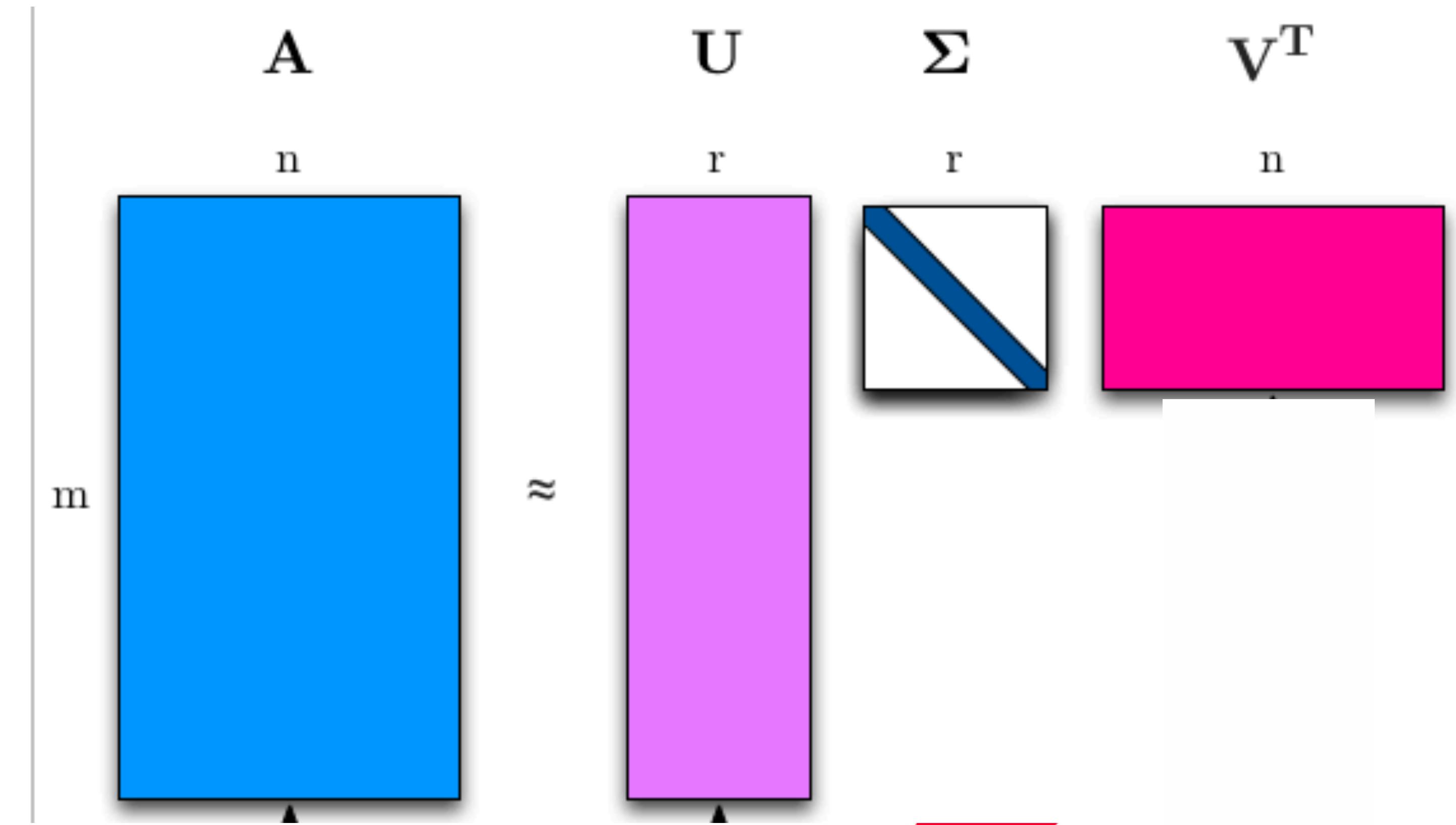
Один из основных недостатков подхода - слишком  
*высокая размерность пространства признаков.*

Вот где актуально снижение  
размерности!



# Понижение размерности

- t-SVD
- PCA
- l1-регуляризация...



# Hashing Trick

**Давайте похешируем элементы исходного  
признакового пространства**



# Hashing Trick

Давайте **похешируем** элементы исходного  
признакового пространства

- Задаем количество bucket'ов K
- Выбираем хеш-функцию, отображающую признак на число [0..K]
- Хешируем каждый признак пространства признаков (размерность пространства N)
- ...
- Profit !

# Hashing Trick

Мы только что сократили размерность пространства признаков  
с  $N$  до  $K$ !



# Hashing Trick

Мы только что сократили размерность пространства признаков  
с  $N$  до  $K$ !



...ПОТРЯСАЮЩЕ

nisovach.ru

$$N \sim 10^5 - 10^7$$

$$K \sim 10^3 - 10^5$$

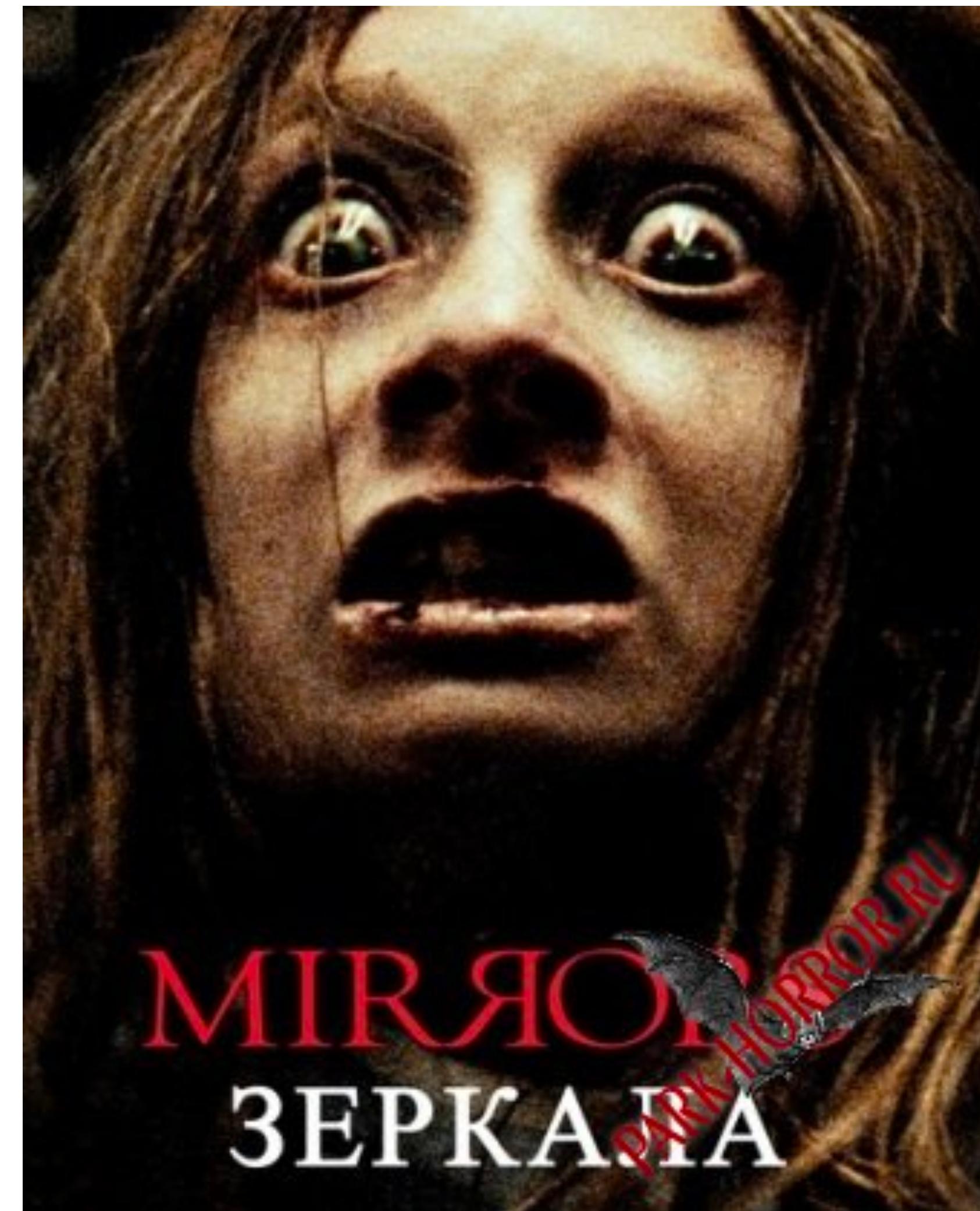
# Locality Sensitive Hashing



# Document similarity

## Зачем нужно?

- определение «зеркал» сайтов



# Document similarity

## Зачем нужно?

- определение «зеркал» сайтов
- антиплагиат



# Document similarity

## Зачем нужно?

- определение «зеркал» сайтов
- антиплагиат
- агрегация похожих новостей

Аэропорт Пулково предложили переименовать в честь Шнурова



РИА Новости вчера в 23:52

Аэропорт Пулково предложили переименовать в честь Шнурова

Поклонники лидера группы "Ленинград" Сергея Шнурова обсуждают возможность назвать в его честь аэропорт Пулково в рамках общенационального конкурса по присвоению имен...



78.ru 08:28

Петербуржцы предложили переименовать Пулково в честь Шнурова

Петербургский аэропорт может получить дополнительное название в честь лидера группы «Ленинград» Сергея Шнурова, сообщается на сайте проекта «Великие имена России».



Gazeta.SPb 00:40

Поклонники «Ленинграда» просят назвать аэропорт именем Шнурова

Но на сайте об этом нет никакой информации Сергей Шнуров отсутствует в списке рассматриваемых великих и выдающихся личностей. В комментариях к посту музыканта многие

# Document similarity

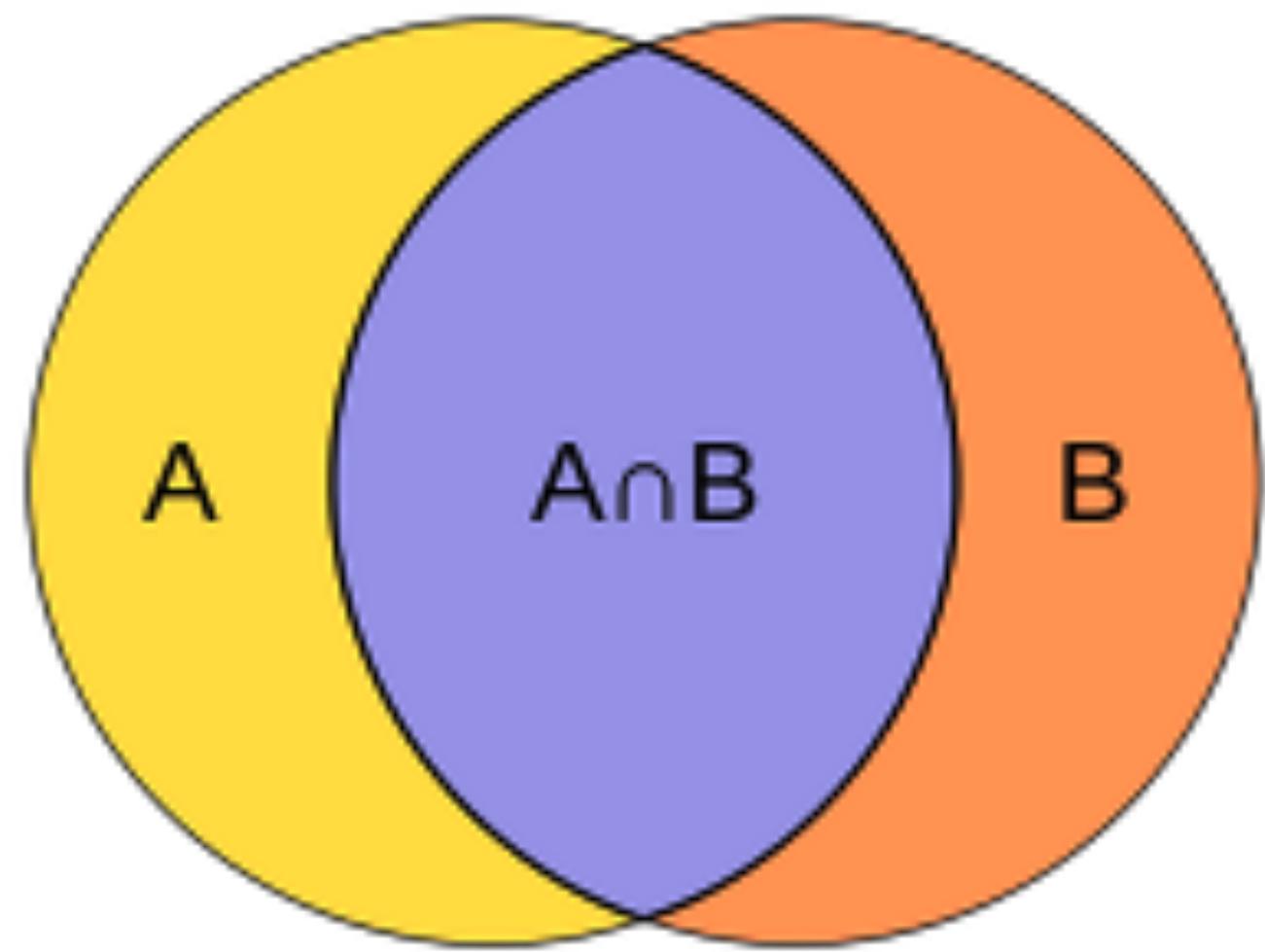
## Зачем нужно?

- определение «зеркал» сайтов
- антиплагиат
- агрегация похожих новостей
- схлопывание дубликатов  
(вакансии)



# Jaccard similarity

- берем документы
- дробим на токены
- считаем коэффициент Жаккара
- определяем дубликаты по порогу
- ...
- Profit!



# Jaccard similarity

**В чем сложность?**

- Необходимо сравнить все пары документов между собой
- Число пар растет квадратично с увеличением корпуса

Идеально было бы выделить группу потенциальных дубликатов, и их уже более детально проверять

# Document shingling\*

Разбиение документа на пересекающиеся последовательности буквенных n-грамм

\*shingle - черепица

# Document shingling\*

Разбиение документа на пересекающиеся последовательности буквенных n-грамм

"однажды в студеную зимнюю пору ...»

k=5

{«однаж», «днажд», «нажды», «ажды », «жды в», «ды в », «ы в с», ...}



\*shingle - черепица

# Document shingling

Все документы разбили на шинглы - буквенные n-граммы

# Document shingling

Все документы разбили на шинглы - буквенные n-граммы

На практике порядок шинглов - 5-10

**Где проблема ?**



# Document shingling

Все документы разбили на шинглы - буквенные n-граммы

На практике порядок шинглов - 5-10

**Где проблема ?**

Шинглов очень много



# Document shingling

Все документы разбили на шинглы - буквенные n-граммы

На практике порядок шинглов - 5-10

**Где проблема ?**

Шинглов очень много

**Что можно сделать ?**



# Document shingling



# Document shingling

Можем представлять документ не множеством шинглов, а множеством хешей от шинглов.

# Document shingling

Можем представлять документ не множеством шинглов, а множеством хешей от шинглов.

- Уменьшим количество шинглов
- Нет необходимости хранить словарь
- Можем вычислять «на лету»
- Можем корректно работать с новыми последовательностями

# Document shingling

Запомним, что обычно мы берем хеши от шинглов, но пока для наглядности будем работать со словами ( шинглы=слова ).

# Document shingling

Запомним, что обычно мы берем хеши от шинглов, но пока для наглядности будем работать со словами ( шинглы=слова ).

## Корпус:

- 1) “Who was the first king of Poland”
- 2) “Who was the first ruler of Poland”
- 3) “Who was the last pharaoh of Egypt”

# Document shingling

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	Who			
2	was			
3	the			
4	first			
5	king			
6	of			
7	Poland			
8	ruler			
9	last			
10	pharaoh			
11	Egypt			

# Document shingling

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	Who	1	1	1
2	was	1	1	1
3	the	1	1	1
4	first	1	1	0
5	king	1	0	0
6	of	1	1	1
7	Poland	1	1	0
8	ruler	0	1	0
9	last	0	0	1
10	pharaoh	0	0	1
11	Egypt	0	0	1

# Document shingling

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	Who	1	1	1
2	was	1	1	1
3	the	1	1	1
4	first	1	1	0
5	king	1	0	0
6	of	1	1	1
7	Poland	1	1	0
8	ruler	0	1	0
9	last	0	0	1
10	pharaoh	0	0	1
11	Egypt	0	0	1

# Document shingling

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	Who	1	1	1
2	was	1	1	1
3	the	1	1	1
4	first	1	1	0
5	king	1	0	0
6	of	1	1	1
7	Poland	1	1	0
8	ruler	0	1	0
9	last	0	0	1
10	pharaoh	0	0	1
11	Egypt	0	0	1

# Document shingling

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	Who	1	1	1
2	was	1	1	1
3	the	1	1	1
4	first	1	1	0
5	king	1	0	0
6	of	1	1	1
7	Poland	1	1	0
8	ruler	0	1	0
9	last	0	0	1
10	pharaoh	0	0	1
11	Egypt	0	0	1

$$J_{12} = \frac{6}{8} = 0.75$$

# Document shingling

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	Who	1	1	1
2	was	1	1	1
3	the	1	1	1
4	first	1	1	0
5	king	1	0	0
6	of	1	1	1
7	Poland	1	1	0
8	ruler	0	1	0
9	last	0	0	1
10	pharaoh	0	0	1
11	Egypt	0	0	1

$$J_{12} = 0.75$$

$$J_{13} = 0.4$$

$$J_{23} = 0.4$$

# Min Hash

Хорошая идея - сделать для каждого документа некую сигнатуру, которые можно быстро сравнивать



# Min Hash

Хорошая идея - сделать для каждого документа некую сигнатуру, которые можно быстро сравнивать

Давайте рандомно переиндексируем словарь и будем каждый документ характеризовать **минимальным индексом слова в нем**

# Min Hash

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	last			
2	Who			
3	Egypt			
4	king			
5	ruler			
6	was			
7	of			
8	Poland			
9	pharaoh			
10	the			
11	first			

# Min Hash

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	last			1
2	Who			
3	Egypt			
4	king			
5	ruler			
6	was			
7	of			
8	Poland			
9	pharaoh			
10	the			
11	first			

# Min Hash

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	last			1
2	Who	2	2	
3	Egypt			
4	king			
5	ruler			
6	was			
7	of			
8	Poland			
9	pharaoh			
10	the			
11	first			

# Min Hash

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	last			1
2	Who	2	2	
3	Egypt			
4	king			
5	ruler			
6	was			
7	of			
8	Poland			
9	pharaoh			
10	the			
11	first			

# Min Hash

Представлять документ индексом всего одного слова - странный план



# Min Hash

Представлять документ индексом всего одного слова - странный план

Давайте повторим процедуру несколько раз:

- 1) Переиндексация словаря (или перемешивание строк)
- 2) Выбираем минимальный id слова в документе в качестве характеристики документа

# Min Hash

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	the			
2	of			
3	Poland			
4	was			
5	first			
6	ruler			
7	Who			
8	Egypt			
9	pharaoh			
10	last			
11	king			

# Min Hash

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	the	1	1	1
2	of			
3	Poland			
4	was			
5	first			
6	ruler			
7	Who			
8	Egypt			
9	pharaoh			
10	last			
11	king			

# Min Hash

index	word	Who was the first king of Poland	Who was the first ruler of Poland	Who was the last pharaoh of Egypt
1	the	1	1	1
2	of			
3	Poland			
4	was			
5	first			
6	ruler			
7	Who			
8	Egypt			
9	pharaoh			
10	last			
11	king			

# Min Hash

Повторим еще 4 раза и выпишем все минхэши для каждого документа:

# Min Hash

Повторим еще 4 раза и выпишем все минхэши для каждого документа:

*MinHashSignature(“Who was the first king of Poland”)* = [2, 1, 1, 2, 1, 1]

*MinHashSignature(“Who was the first ruler of Poland”)* = [2, 1, 1, 1, 1, 1]

*MinHashSignature(“Who was the last pharaoh of Egypt”)* = [1, 1, 3, 4, 4, 1]

# Min Hash

Повторим еще 4 раза и выпишем все минхэши для каждого документа:

*MinHashSignature("Who was the first king of Poland") = [2, 1, 1, 2, 1, 1]*

*MinHashSignature("Who was the first ruler of Poland") = [2, 1, 1, 1, 1, 1]*

*MinHashSignature("Who was the last pharaoh of Egypt") = [1, 1, 3, 4, 4, 1]*

Можем посчитать коэффициент Жаккара между минхэш-сигнатурами:

$$J_{12}^{minhash} = \frac{5}{6} \approx 0.83$$

$$J_{13}^{minhash} = \frac{2}{6} \approx 0.33$$

$$J_{23}^{minhash} = \frac{2}{6} \approx 0.33$$

# Min Hash

Повторим еще 4 раза и выпишем все минхэши для каждого документа:

*MinHashSignature("Who was the first king of Poland") = [2, 1, 1, 2, 1, 1]*

*MinHashSignature("Who was the first ruler of Poland") = [2, 1, 1, 1, 1, 1]*

*MinHashSignature("Who was the last pharaoh of Egypt") = [1, 1, 3, 4, 4, 1]*

Можем посчитать коэффициент Жаккара между минхэш-сигнатурами:

$$J_{12}^{minhash} = \frac{5}{6} \approx 0.83$$

$$J_{13}^{minhash} = \frac{2}{6} \approx 0.33$$

$$J_{23}^{minhash} = \frac{2}{6} \approx 0.33$$

$$J_{12} = 0.75$$

$$J_{13} = 0.4$$

$$J_{23} = 0.4$$

# Min Hash

Повторим еще 4 раза и выпишем все минхэши для каждого документа:

$\text{MinHashSignature}(\text{"Who was the first king of Poland"}) = [2, 1, 1, 2, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the first ruler of Poland"}) = [2, 1, 1, 1, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the last pharaoh of Egypt"}) = [1, 1, 3, 4, 4, 1]$

Можем посчитать коэффициент Жаккара между минхэш-сигнатурами:

$$J_{12}^{\text{minhash}} = \frac{5}{6} \approx 0.83$$

$$J_{12} = 0.75$$

$$J_{13}^{\text{minhash}} = \frac{2}{6} \approx 0.33$$

$$J_{13} = 0.4$$

$$J_{23}^{\text{minhash}} = \frac{2}{6} \approx 0.33$$

$$J_{23} = 0.4$$

# Min Hash

Какова вероятность, что для пары документов пара минхэшей совпадет?



# Min Hash

Какова вероятность, что для пары документов пара минхэшей совпадет?

Вероятность совпадения равна **коэффициенту Жаккара** между этими документами.

# Min Hash

Какова вероятность, что для пары документов пара минхэшей совпадет?

Вероятность совпадения равна **коэффициенту Жаккара** между этими документами.

Коэффициент Жаккара между минхэш-сигнатурами - оценка «честного» коэффициента Жаккара между шинглированными документами

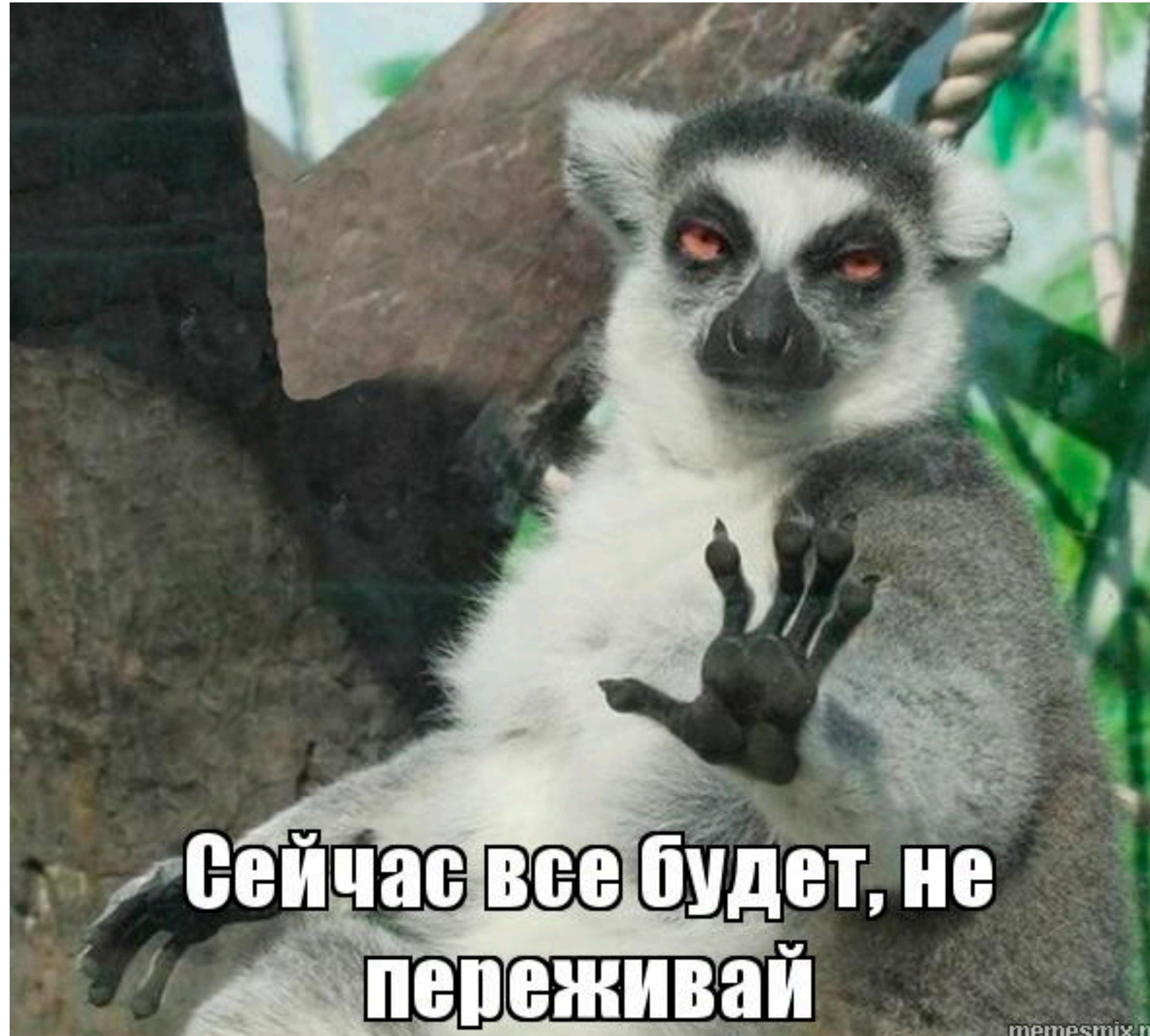
Чем больше операций переиндексации мы делаем  
- тем точнее оценка

# Min Hash

Окей, алгоритм называется MinHash, но где здесь хэш?

# Min Hash

Окей, алгоритм называется MinHash, но где здесь хэш?



# Min Hash

Окей, алгоритм называется MinHash, но где здесь хэш?

Перемешивание словаря не очень эффективно:

- надо хранить словарь
- надо хранить перестановки

# Min Hash

Окей, алгоритм называется MinHash, но где здесь хэш?

Перемешивание словаря не очень эффективно:

- надо хранить словарь
- надо хранить перестановки

**В чем идейно смысл перемешивания словаря?**

# Min Hash

Окей, алгоритм называется MinHash, но где здесь хэш?

Перемешивание словаря не очень эффективно:

- надо хранить словарь
- надо хранить перестановки

**В чем идейно смысл перемешивания словаря?**

В сопоставлении слову некоего индекса

# Min Hash

Давайте вместо перемешивания словаря каждый раз:

- выбирать хеш-функцию (каждый раз разную)
- применять ее на лету ко всем шинглам документов
- на данной итерации документ характеризовать минимальным хешем из получившихся по документу

# Min Hash

И это работает ровно точно так же, как и раньше, только эффективнее:



# Min Hash

И это работает ровно точно так же, как и раньше, только эффективнее:

- не надо хранить словарь
- можно работать с новыми элементами вне словаря («out of vocabulary»)
- храним только набор хеш-функций

# Min Hash

Итак:

- мы научились эффективно строить минхэш-сигнатуры для каждого из документов
- выяснили, что коэффициент Жаккара для минхэш-сигнатур - аппроксимация честного коэффициента Жаккара по шинглам

# Min Hash

Итак:

- мы научились эффективно строить минхэш-сигнатуры для каждого из документов
- выяснили, что коэффициент Жаккара для минхэш-сигнатур - аппроксимация честного коэффициента Жаккара по шинглам

Мы молодцы!

Но пар документов все так же много...

# Locality Sensitive Hashing

Давайте еще разок посмотрим на получившиеся сигнатуры:

# Locality Sensitive Hashing

Давайте еще разок посмотрим на получившиеся сигнатуры:

$\text{MinHashSignature}(\text{"Who was the first king of Poland"}) = [2, 1, 1, 2, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the first ruler of Poland"}) = [2, 1, 1, 1, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the last pharaoh of Egypt"}) = [1, 1, 3, 4, 4, 1]$

# Locality Sensitive Hashing

Давайте еще разок посмотрим на получившиеся сигнатуры:

$\text{MinHashSignature}(\text{"Who was the first king of Poland"}) = [2, 1, 1, 2, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the first ruler of Poland"}) = [2, 1, 1, 1, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the last pharaoh of Egypt"}) = [1, 1, 3, 4, 4, 1]$

**Кажется, тут есть кое-что интересное...**

# Locality Sensitive Hashing

Давайте еще разок посмотрим на получившиеся сигнатуры:

$\text{MinHashSignature}(\text{"Who was the first king of Poland"}) = [2, 1, 1, 2, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the first ruler of Poland"}) = [2, 1, 1, 1, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the last pharaoh of Egypt"}) = [1, 1, 3, 4, 4, 1]$

Кажется, тут есть кое-что интересное...

# Locality Sensitive Hashing

Давайте еще разок посмотрим на получившиеся сигнатуры:

$\text{MinHashSignature}(\text{"Who was the first king of Poland"}) = [2, 1, 1, 2, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the first ruler of Poland"}) = [2, 1, 1, 1, 1, 1]$

$\text{MinHashSignature}(\text{"Who was the last pharaoh of Egypt"}) = [1, 1, 3, 4, 4, 1]$

**Кажется, тут есть кое-что интересное...**

Давайте разобьем полученные  
сигнатуры на 2 группы по 3 элемента

# Locality Sensitive Hashing

MinHashSignature(*“Who was the first king of Poland”*) = [2, 1, 1, 2, 1, 1] = **[211, 211]**

MinHashSignature(*“Who was the first ruler of Poland”*) = [2, 1, 1, 1, 1, 1] = **[211, 111]**

MinHashSignature(*“Who was the last pharaoh of Egypt”*) = [1, 1, 3, 4, 4, 1] = **[113, 441]**

# Locality Sensitive Hashing

MinHashSignature(*“Who was the first king of Poland”*) = [2, 1, 1, 2, 1, 1] = [**211, 211**]

MinHashSignature(*“Who was the first ruler of Poland”*) = [2, 1, 1, 1, 1, 1] = [**211, 111**]

MinHashSignature(*“Who was the last pharaoh of Egypt”*) = [1, 1, 3, 4, 4, 1] = [**113, 441**]

Заметим, что околодубликаты имеют общую группу

# Locality Sensitive Hashing

MinHashSignature(*“Who was the first king of Poland”*) = [2, 1, 1, 2, 1, 1] = [**211, 211**]

MinHashSignature(*“Who was the first ruler of Poland”*) = [2, 1, 1, 1, 1, 1] = [**211, 111**]

MinHashSignature(*“Who was the last pharaoh of Egypt”*) = [1, 1, 3, 4, 4, 1] = [**113, 441**]

Заметим, что околодубликаты имеют общую группу

Давайте оценим вероятность того, что для похожих документов будет хотя бы одна общая группа

Проведем оценку для пары документов 1 и 2

# Locality Sensitive Hashing

1) Вероятность совпадения любой пары минхэшей:

# Locality Sensitive Hashing

1) Вероятность совпадения любой пары минхэшей:

$$P_1 = J_{12} = 0.75$$

# Locality Sensitive Hashing

1) Вероятность совпадения **любой пары** минхэшей:

$$P_1 = J_{12} = 0.75$$

2) Вероятность совпадения **всех минхэшей** в группе из 3x элементов:

# Locality Sensitive Hashing

1) Вероятность совпадения **любой пары** минхэшей:

$$P_1 = J_{12} = 0.75$$

2) Вероятность совпадения **всех минхэшей** в группе из 3x элементов:

$$P_2 = P_1^3 = J_{12}^3 = 0.75^3 \approx 0.42$$

# Locality Sensitive Hashing

1) Вероятность совпадения **любой пары** минхэшей:

$$P_1 = J_{12} = 0.75$$

2) Вероятность совпадения **всех минхэшей** в группе из 3x элементов:

$$P_2 = P_1^3 = J_{12}^3 = 0.75^3 \approx 0.42$$

3) Вероятность того, что **хотя бы одна пара** минхэшей в группе из 3x элементов **не совпадет**:

# Locality Sensitive Hashing

1) Вероятность совпадения **любой пары** минхэшей:

$$P_1 = J_{12} = 0.75$$

2) Вероятность совпадения **всех минхэшей** в группе из 3x элементов:

$$P_2 = P_1^3 = J_{12}^3 = 0.75^3 \approx 0.42$$

3) Вероятность того, что **хотя бы одна пара** минхэшей в группе из 3x элементов **не совпадет**:

$$P_3 = 1 - P_2 = 1 - J_{12}^3 \approx 0.58$$

# Locality Sensitive Hashing

4) Вероятность п.3 сразу в обоих группах по 3 элемента:

# Locality Sensitive Hashing

4) Вероятность п.3 сразу в обоих группах по 3 элемента:

$$P_4 = P_3^2 = (1 - J_{12}^3)^2 \approx 0.58^2 = 0.34$$

# Locality Sensitive Hashing

4) Вероятность п.3 сразу в обоих группах по 3 элемента:

$$P_4 = P_3^2 = (1 - J_{12}^3)^2 \approx 0.58^2 = 0.34$$

5) Вероятность того, что **хотя бы в одной группе по 3 элемента минхэши совпадут**:

# Locality Sensitive Hashing

4) Вероятность п.3 сразу в обоих группах по 3 элемента:

$$P_4 = P_3^2 = (1 - J_{12}^3)^2 \approx 0.58^2 = 0.34$$

5) Вероятность того, что **хотя бы в одной группе по 3 элемента минхэши совпадут**:

$$P_5 = 1 - P_4 = 1 - (1 - J_{12}^3)^2 \approx 0.66$$

# Locality Sensitive Hashing

4) Вероятность п.3 сразу в обоих группах по 3 элемента:

$$P_4 = P_3^2 = (1 - J_{12}^3)^2 \approx 0.58^2 = 0.34$$

5) Вероятность того, что **хотя бы в одной группе по 3 элемента минхэши совпадут**:

$$P_5 = 1 - P_4 = 1 - (1 - J_{12}^3)^2 \approx 0.66$$

Вероятность, что документы 1 и 3 - дубликаты:

$$P_{sim} = 1 - (1 - J_{13}^3)^2 = 1 - (1 - 0.4^3)^2 \approx 0.12$$

# Locality Sensitive Hashing

В общем случае:

Разбили сигнатуру на **b** групп по **n** элементов в каждой.

# Locality Sensitive Hashing

В общем случае:

Разбили сигнатуру на **b** групп по **n** элементов в каждой.

Вероятность того, что документы имеют хотя бы одну общую группу:

$$P_{sim} = 1 - (1 - J^n)^b$$

# Locality Sensitive Hashing

В общем случае:

Разбили сигнатуру на **b** групп по **n** элементов в каждой.

Вероятность того, что документы имеют хотя бы одну общую группу:

$$P_{sim} = 1 - (1 - J^n)^b$$

*n* - количество элементов в каждой группе

*b* - количество групп разбиения

*J* - коэффициент Жаккара между документами

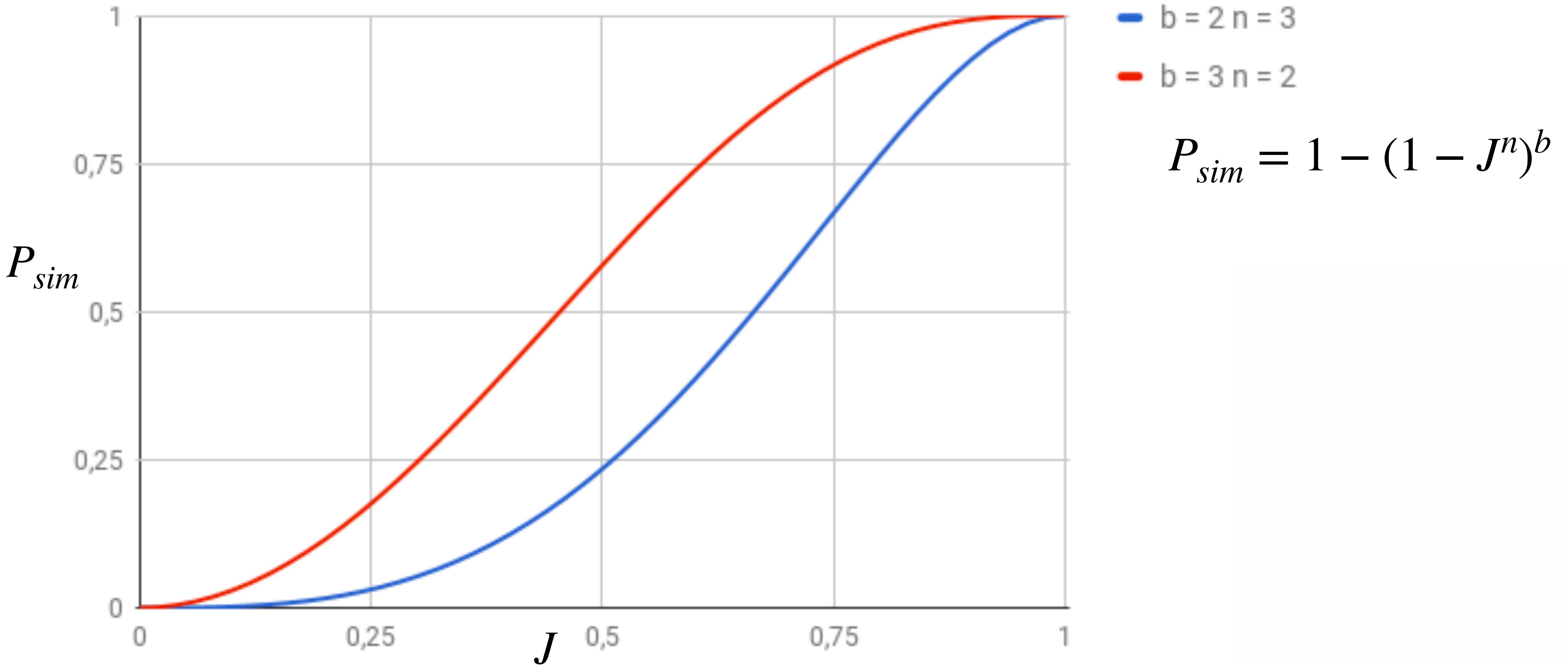
$P_{sim}$  - вероятность совпадения хотя бы 1 общей группы

# Locality Sensitive Hashing

Мы могли бы разбить сигнатуру на 3 группы по 2 элемента:

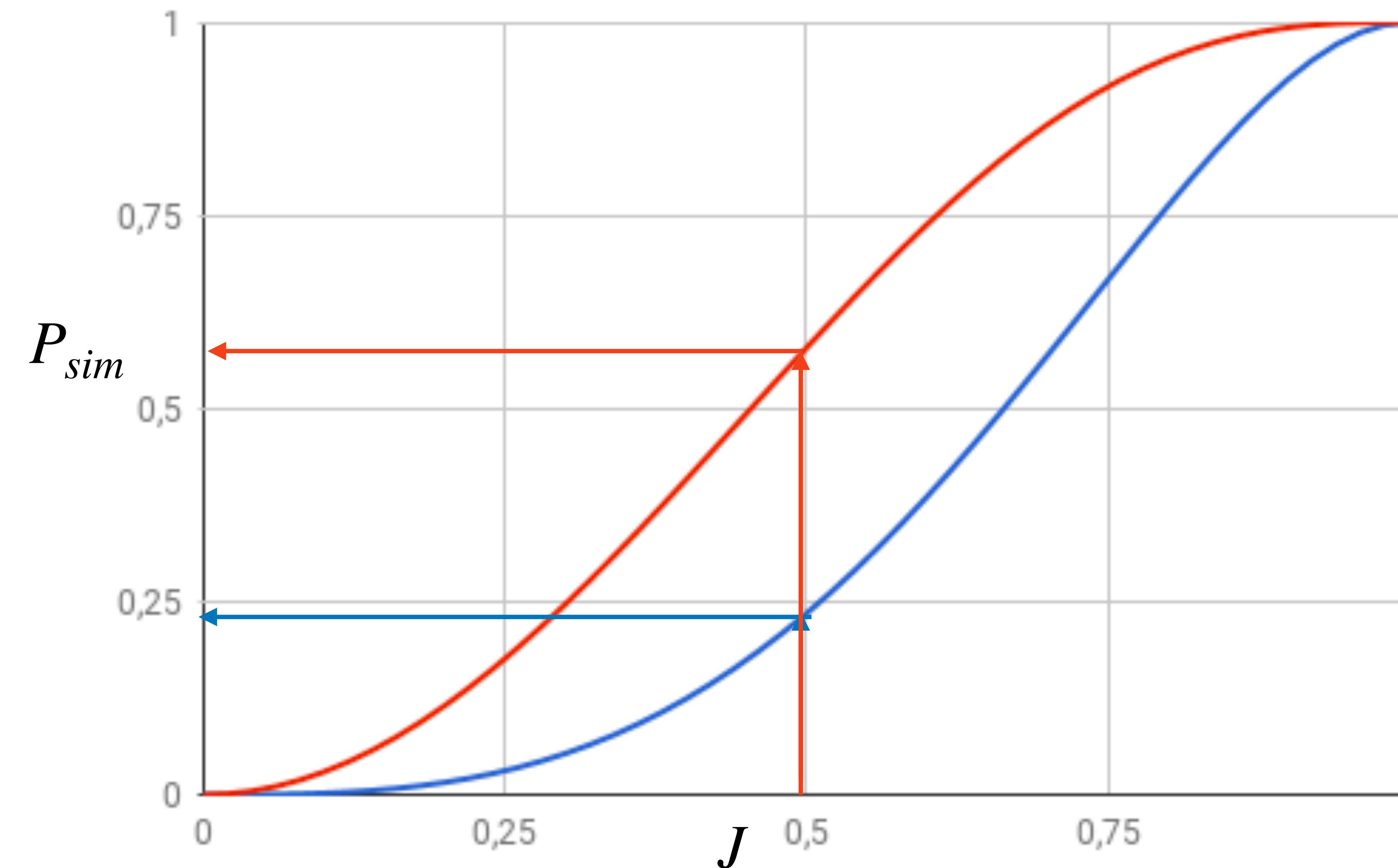
# Locality Sensitive Hashing

Мы могли бы разбить сигнатуру на 3 группы по 2 элемента:



# Locality Sensitive Hashing

Мы могли бы разбить сигнатуру на 3 группы по 2 элемента:



- $b = 2 n = 3$
- $b = 3 n = 2$

$$P_{sim} = 1 - (1 - J^n)^b$$

Чем больше размер группы - тем точнее склеиваем дубликаты, но теряем в покрытии

$b \uparrow \Rightarrow fp \downarrow ; fn \uparrow$

$b \downarrow \Rightarrow fp \uparrow ; fn \downarrow$

# Locality Sensitive Hashing

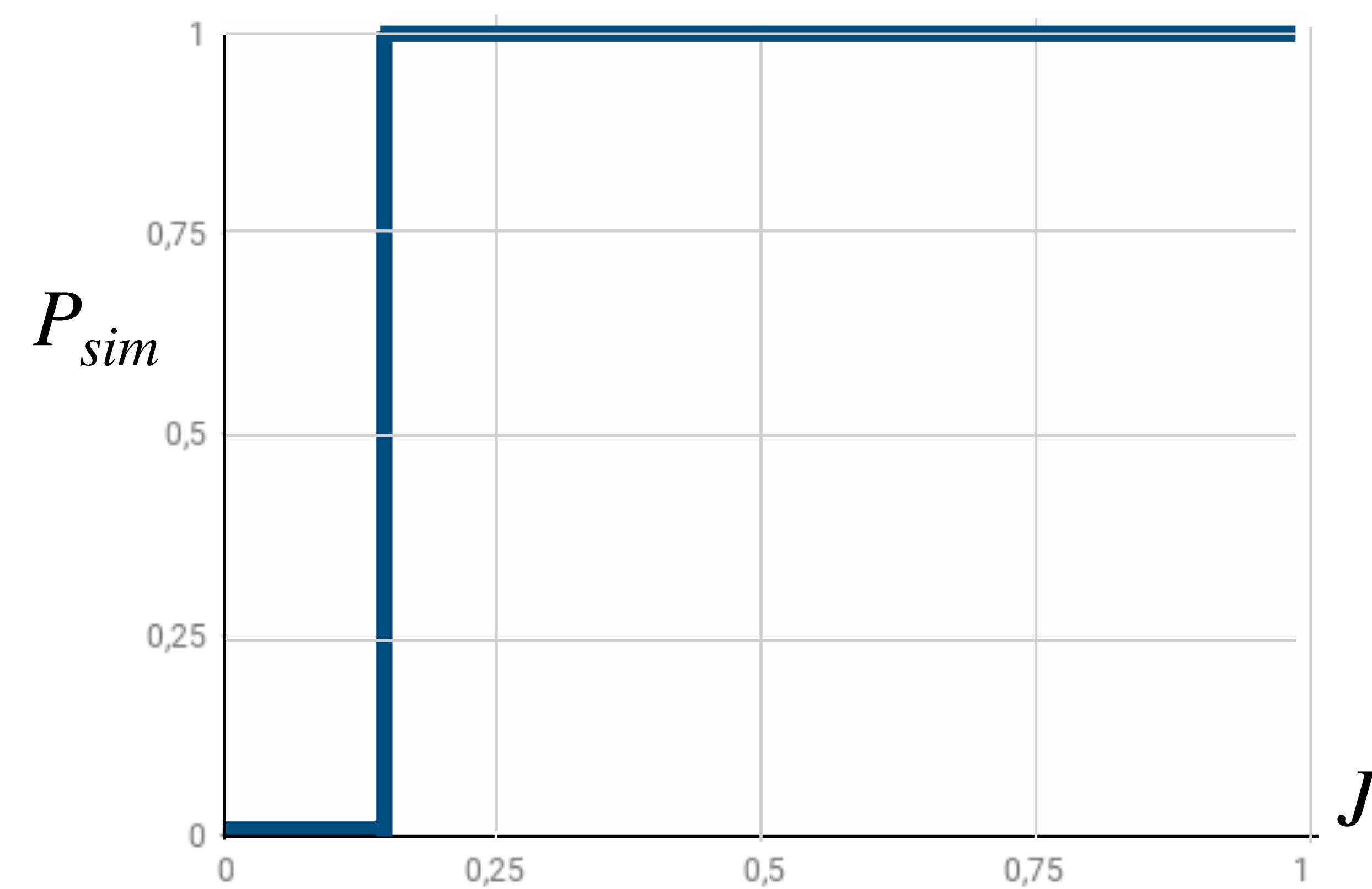
Обычно задаются пороговым значением коэффициента Жаккара  $t$ , свыше которого считают документы дубликатами.

Затем подбирают количество групп разбиения и размер группы.

# Locality Sensitive Hashing

Обычно задаются пороговым значением коэффициента Жаккара  $t$ , свыше которого считают документы дубликатами.

Затем подбирают количество групп разбиения и размер группы.



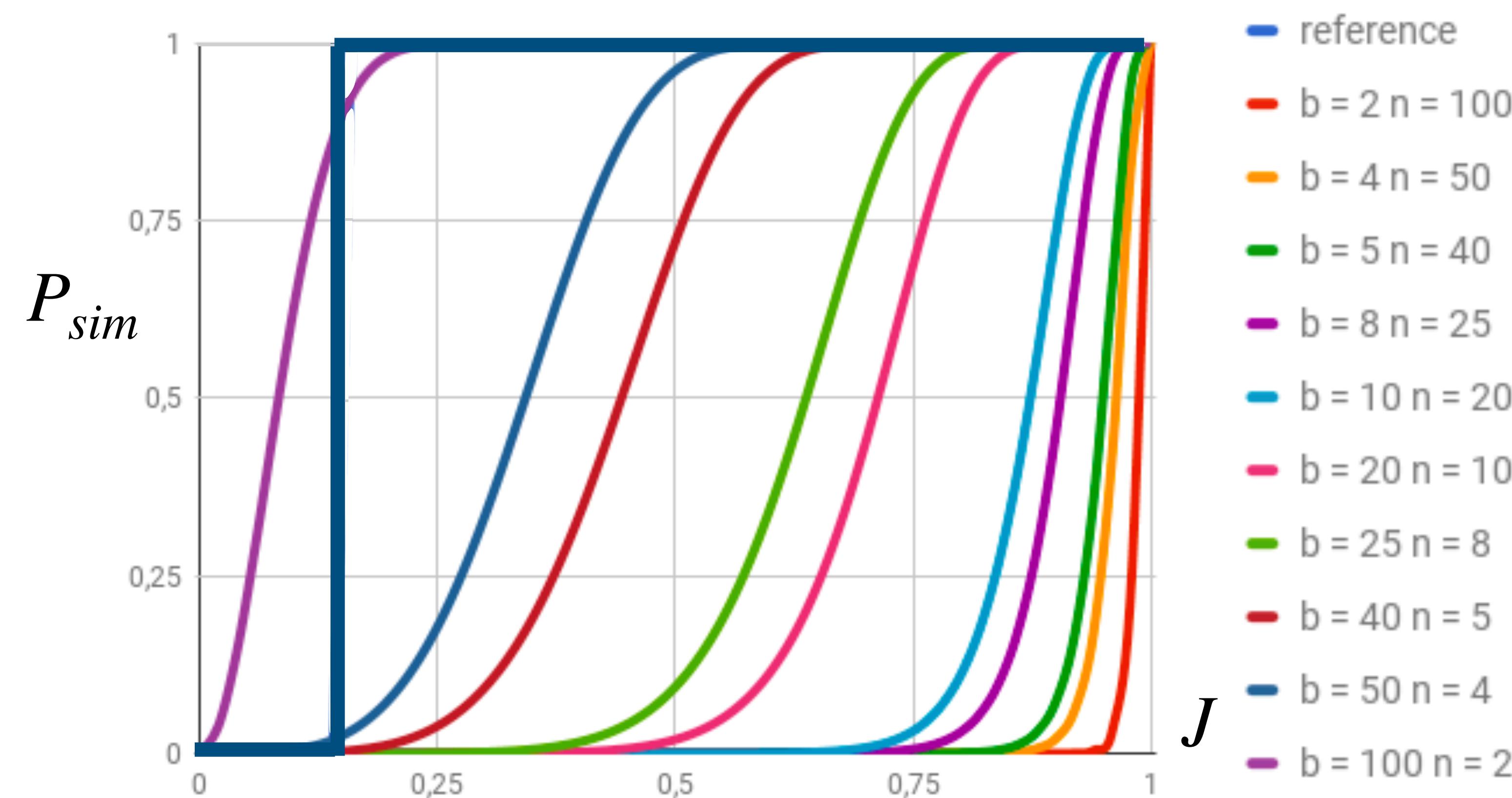
$$b \cdot n = const$$

$$t \approx \frac{1}{b^{\frac{1}{n}}}$$

# Locality Sensitive Hashing

Обычно задаются пороговым значением коэффициента Жаккара  $t$ , свыше которого считают документы дубликатами.

Затем подбирают количество групп разбиения и размер группы.



$$b \cdot n = const$$

$$t \approx \frac{1}{b}$$

# Locality Sensitive Hashing

Итак:

- мы научились эффективно строить минхэш-сигнатуры для каждого из документов
- выяснили, что коэффициент Жаккара для минхэш-сигнатур - аппроксимация честного коэффициента Жаккара по шинглам
- разбили минхэш-сигнатуры на группы
- посчитали вероятность того, что документы, имеющие общую группу - дубликаты

# Locality Sensitive Hashing

MinHashSignature("Who was *the first king of Poland*") = [2, 1, 1, 2, 1, 1] = [**211, 211**]

MinHashSignature("Who was *the first ruler of Poland*") = [2, 1, 1, 1, 1, 1] = [**211, 111**]

MinHashSignature("Who was *the last pharaoh of Egypt*") = [1, 1, 3, 4, 4, 1] = [**113, 441**]

# Locality Sensitive Hashing

MinHashSignature("Who was the first king of Poland") = [2, 1, 1, 2, 1, 1] = [211, 211]

MinHashSignature("Who was the first ruler of Poland") = [2, 1, 1, 1, 1, 1] = [211, 111]

MinHashSignature("Who was the last pharaoh of Egypt") = [1, 1, 3, 4, 4, 1] = [113, 441]

Не одно и то же

# Locality Sensitive Hashing

MinHashSignature("Who was the first king of Poland") = [2, 1, 1, 2, 1, 1] = [211, 211]

MinHashSignature("Who was the first ruler of Poland") = [2, 1, 1, 1, 1, 1] = [211, 111]

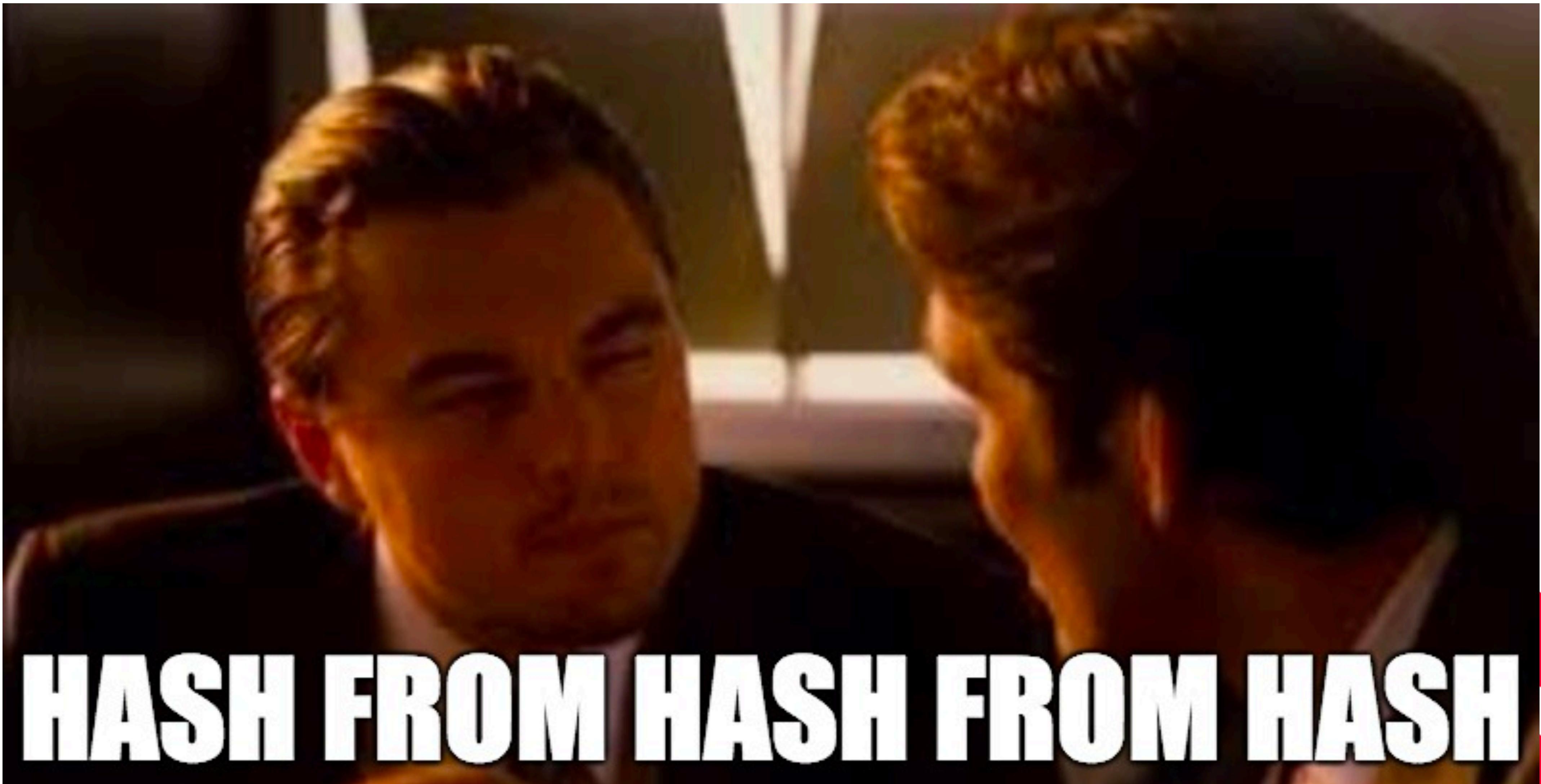
MinHashSignature("Who was the last pharaoh of Egypt") = [1, 1, 3, 4, 4, 1] = [113, 441]

Группы минхэшей называются супершинглами.

Не одно и то же

Обычно сами супершинглы еще раз хешируют с учетом расположения и получают ключи для хэш-таблицы, в которой будут лежать потенциальные дубликаты.

# Locality Sensitive Hashing



# Locality Sensitive Hashing

Упрощенно все выглядит так:

```
aaa= hash(111) => id("Who was the first ruler of Poland"),  
bbb = hash(211) => id("Who was the first king of Poland")  
ccc = hash(113) => id("Who was the last pharaoh of Egypt")  
ddd = hash(441) => id("Who was the last pharaoh of Egypt")  
...
```

Таблица поиска дубликатов

# Locality Sensitive Hashing

**Как искать дубликаты/похожие документы:**

- Приходящий запрос обрабатывается тем же самым образом, что и корпус: шинглирование, формирование минхэш-сигнатуры **теми же хэш-функциями**

# Locality Sensitive Hashing

## Как искать дубликаты/похожие документы:

- Приходящий запрос обрабатывается тем же самым образом, что и корпус: шинглирование, формирование минхэш-сигнатуры **теми же хэш-функциями**
- Происходит разбивка сигнатуры запроса на супершинглы

# Locality Sensitive Hashing

## Как искать дубликаты/похожие документы:

- Приходящий запрос обрабатывается тем же самым образом, что и корпус: шинглирование, формирование минхэш-сигнатуры **теми же хэш-функциями**
- Происходит разбивка сигнатуры запроса на супершинглы
- Используем хэши от супершинглов в качестве ключей и ищем в словаре дубликатов элементы с такими ключами

# Locality Sensitive Hashing

## Как искать дубликаты/похожие документы:

- Приходящий запрос обрабатывается тем же самым образом, что и корпус: шинглирование, формирование минхэш-сигнатуры **теми же хэш-функциями**
- Происходит разбивка сигнатуры запроса на супершинглы
- Используем хэши от супершинглов в качестве ключей и ищем в словаре дубликатов элементы с такими ключами
- Если нашли - с большой вероятностью это дубликаты/похожие документы

# Locality Sensitive Hashing

После нахождения потенциальных дубликатов можно их проверить с помощью честного коэффициента Жаккара, так как это уже не настолько затратно.

Данная постпроверка позволит избавиться от ложноположительных результатов («склеились» как дубликаты, но дубликатами не являются)

# SimHash

MinHash - не единственный способ сформировать сигнатуру документа. Рассмотрим еще один из способов - SimHash.

# SimHash

MinHash - не единственный способ сформировать сигнатуру документа. Рассмотрим еще один из способов - SimHash.

- SimHash особым образом формирует сигнатуру документа
- сигнатура затем разбивается на супершинглы
- супершинглы хешируют и используют в качестве ключей для поиска дубликатов

# SimHash

- Сформируем шинглы документа

# SimHash

- Сформируем шинглы документа
- Затем для каждого шингла сформируем n-битный бинарный хэш ( вектор из 0 и 1)

# SimHash

- Сформируем шинглы документа
- Затем для каждого шингла сформируем n-битный бинарный хэш ( вектор из 0 и 1)
- Инвертируем 0 в -1

# SimHash

- Сформируем шинглы документа
- Затем для каждого шингла сформируем n-битный бинарный хэш ( вектор из 0 и 1)
- Инвертируем 0 в -1
- Взвесим бинарные хеши с весами TF для каждого шингла и сложим

# SimHash

- Сформируем шинглы документа
- Затем для каждого шингла сформируем n-битный бинарный хэш ( вектор из 0 и 1)
- Инвертируем 0 в -1
- Взвесим бинарные хеши с весами TF для каждого шингла и сложим
- Бинаризуем полученный суммарный хэш

# SimHash

- Сформируем шинглы документа
- Затем для каждого шингла сформируем n-битный бинарный хэш ( вектор из 0 и 1)
- Инвертируем 0 в -1
- Взвесим бинарные хеши с весами TF для каждого шингла и сложим
- Бинаризуем полученный суммарный хэш
- Это и есть SimHash сигнатура

# SimHash



# SimHash

о боже мама мама я схожу с ума



# SimHash

о боже мама мама я схожу с ума

о

боже

мама

я

схожу

с

ума



# SimHash

о боже мама мама я схожу с ума

о 01010001

боже 11101100

мама 00110011

я 10111101

схожу 00001001

с 00011010

ума 10111100

8-bit hash

# SimHash

о боже мама мама я схожу с ума

о	01010001
боже	11101100
мама	00110011 <b>x 2</b>
я	10111101
схожу	00001001
с	00011010
ума	10111100
	8-bit hash

# SimHash

о боже мама мама я схожу с ума

о	01010001
боже	11101100
мама	00110011 <b>x 2</b> [ ; ; ; ; ; ; ]
я	10111101
схожу	00001001
с	00011010
ума	10111100

$$-1+1-2+1-1-1+1=-2$$

# SimHash

о боже мама мама я схожу с ума

о	01010001
боже	11101100
мама	00110011 <b>x 2</b>
я	10111101
схожу	00001001
с	00011010
ума	10111100

-1+1-2+1-1-1+1=-2

[ -2; ; ; ; ; ; ]



# SimHash

о боже мама мама я схожу с ума

о  
боже  
мама  
я  
схожу  
с  
ума

01010001
11101100
00110011 <b>x 2</b>
10111101
00001001
00011010
10111100

[ -2; -4; ; ; ; ; ; ]



-4

# SimHash

о боже мама мама я схожу с ума

о  
боже  
мама  
я  
схожу  
с  
ума

01010001  
11101100  
00110011 **x 2**  
10111101  
00001001  
00011010  
10111100

[ -2; -4; 2; ; ; ; ; ]

2

# SimHash

о боже мама мама я схожу с ума

о  
боже  
мама  
я  
схожу  
с  
ума

01010001  
11101100  
00110011    x 2  
10111101  
00001001  
00011010  
10111100

[ -2; -4; 2; 4; ; ; ; ]

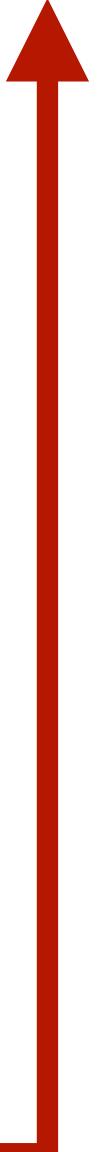
4

# SimHash

о боже мама мама я схожу с ума

о	01010001	
боже	11101100	
мама	00110011	$\times 2$
я	10111101	
схожу	00001001	
с	00011010	
ума	10111100	
	2	

[ -2; -4; 2; 4; 2; ; ; ]

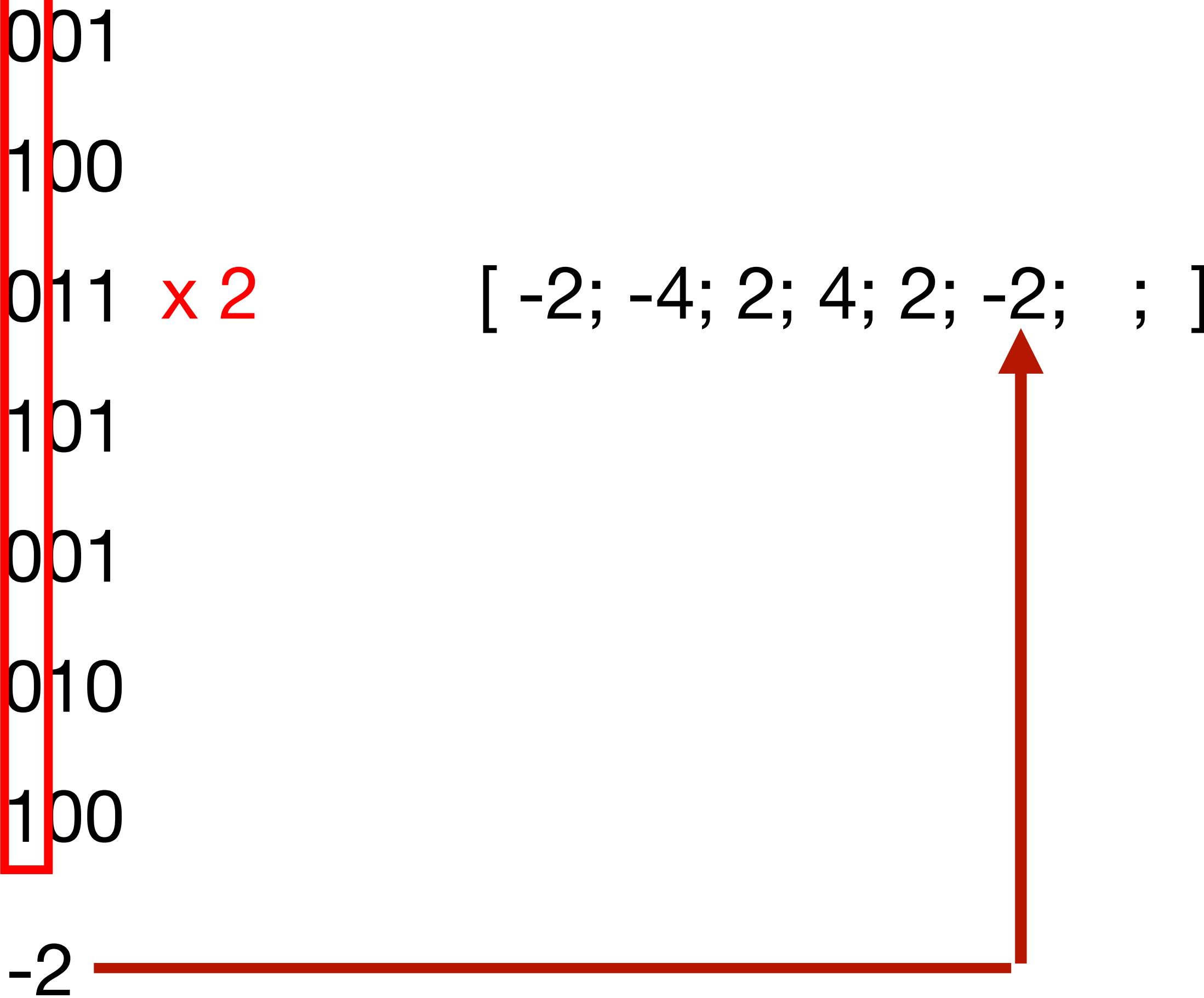


# SimHash

о боже мама мама я схожу с ума

о	01010001	
боже	11101100	
мама	00110011	$\times 2$
я	10111101	
схожу	00001001	
с	00011010	
ума	10111100	

-2



# SimHash

о боже мама мама я схожу с ума

о	01010001
боже	11101100
мама	00110011 <b>x 2</b>
я	10111101
схожу	00001001
с	00011010
ума	10111100

[ -2; -4; 2; 4; 2; -2; -2; ]

-2

# SimHash

о боже мама мама я схожу с ума

о	01010001
боже	11101100
мама	00110011 <span style="color:red">x 2</span>
я	10111101
схожу	00001001
с	00011010
ума	10111100

2

[ -2; -4; 2; 4; 2; -2; -2; 2 ]

# SimHash

о боже мама мама я схожу с ума

о	01010001	
боже	11101100	
мама	00110011	<b>x 2</b> [ -2; -4; 2; 4; 2; -2; -2; 2 ]
я	10111101	
схожу	00001001	
с	00011010	
ума	10111100	

# SimHash

о боже мама мама я схожу с ума

о	01010001	
боже	11101100	
мама	00110011 <b>x 2</b>	[ -2; -4; 2; 4; 2; -2; -2; 2 ]
я	10111101	
схожу	00001001	
с	00011010	
ума	10111100	



binarization

[ 0; 0; 1; 1; 1; 0; 0; 1 ]

# SimHash

о боже мама мама я схожу с ума

о	01010001	
боже	11101100	
мама	00110011 <b>x 2</b>	[ -2; -4; 2; 4; 2; -2; -2; 2 ]
я	10111101	
схожу	00001001	
с	00011010	
ума	10111100	

[ 0; 0; 1; 1; 1; 0; 0; 1 ]

SimHash сигнатура

# SimHash

о боже мама мама я схожу с ума

о	01010001	
боже	11101100	
мама	00110011 <b>x 2</b>	[ -2; -4; 2; 4; 2; -2; -2; 2 ]
я	10111101	
схожу	00001001	
с	00011010	[ 0; 0; 1; 1; 1; 0; 0; 1 ]
ума	10111100	

Супершинглы

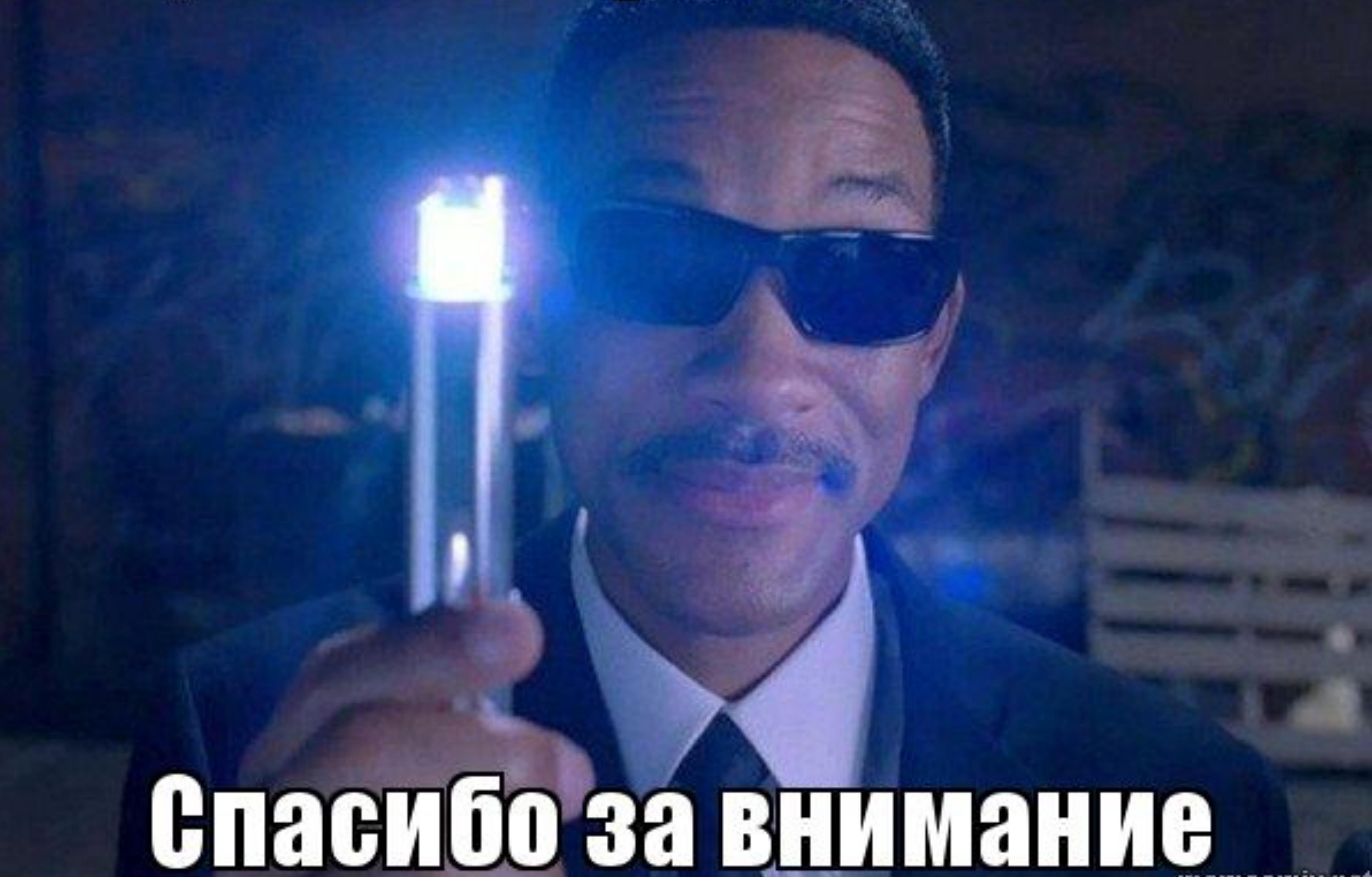
# LSH - ограничения

Работает только для сравнительно больших документов, так как нужно достаточно количество шинглов

Есть более эффективные структуры данных для поиска дубликатов. Например, LSH-forest, который сразу позволяет найти top-k наиболее похожих кандидатов

@mail.

# Презентация окончена



## Спасибо за внимание