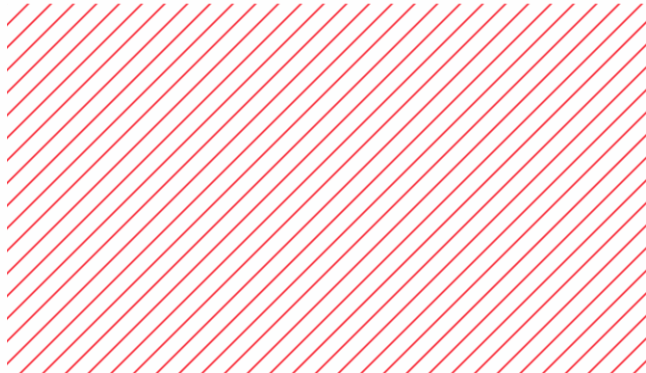
академия больших данных

mail.ru group

# Approximate алгоритмы для больших данных

Дмитрий Киселёв



# План

- Хэш и примеры применений
  - Извлечение признаков
  - Разделение на группы
  - Фильтрация
  - Расчет статистик
- Поиск ближайших соседей
  - Примерный подсчет расстояния
  - Примерный поиск ближайших соседей

# Что такое хэш

# Хэш

Хэш функция переводит вход (число / строку / whatever) в целочисленный номер корзины (bucket number)

Хороший хэш должен равномерно распределять по корзинам ключи.

Хэш от одного ключа всегда попадает в одну корзину

Пример: остаток от деления на простое число для целочисленных ключей

Пример: конвертируем символ для строки в Int (Unicode / ASCII), суммируем.

# Hashing trick

Аналог One-Hot-encoding

Рассмотрим фичу с именем “country”

1. Создаем хэш-таблицу с весами линейной модели
2. Пусть для текущей строки ее значение “russia”
3. Берем хэш от “country_russia” и достаем из хэш-таблицы соответствующий вес

HashingTF в Spark

1. Вместо сохранения конкретных слов, берем хэши от них
2. Сохраняем это в хэш-таблицу (хэш слова -> встречаемость)

# A/B/n-тесты

- Берем id пользователя – превращаем в строку
- Конкатенируем с солью (например, название текущего теста)
- Берем от этого хэш (говорят, что MD5 лучший вариант)
- Массив байт конвертируем в BigInt
- Делим на самый большой BigInt – получаем значение CDF равномерного распределения
- Сравниваем CDF с границами вариаций

# Bloom Filter

Фильтрует данные на вхождение в множество

Состоит из массива $m$ бит

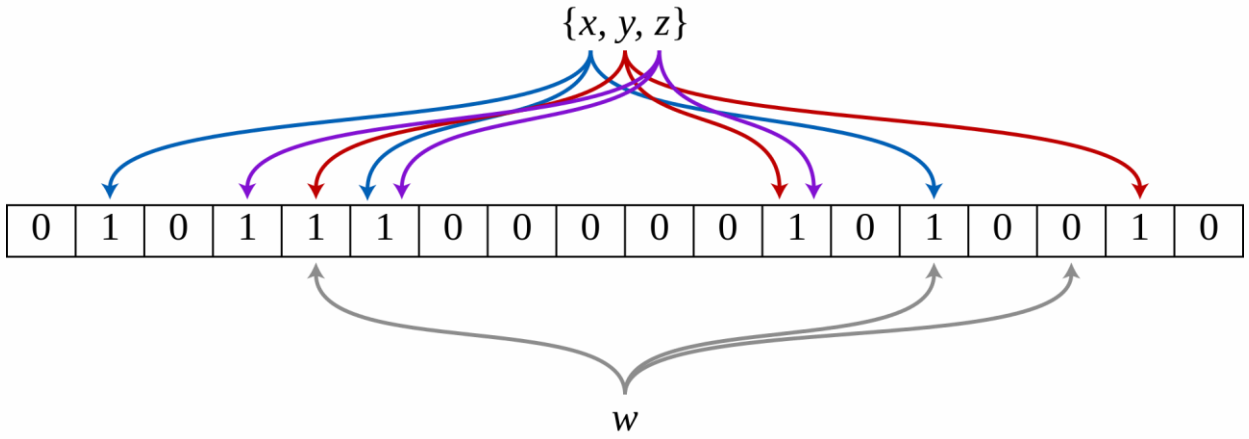
Берем $k$ хэш функций, которые кладут в одну из $m$ корзин и заполняем нулевые значения по имеющимся данным

Если при запросе одна из функций попадет в нулевую корзину, то таких данных точно нет в базе

Если p – желаемая вероятность ошибки, а n – количество данных, то

$m \approx -1.44\, n \log_2 p$

$k \approx -\log_2 p$



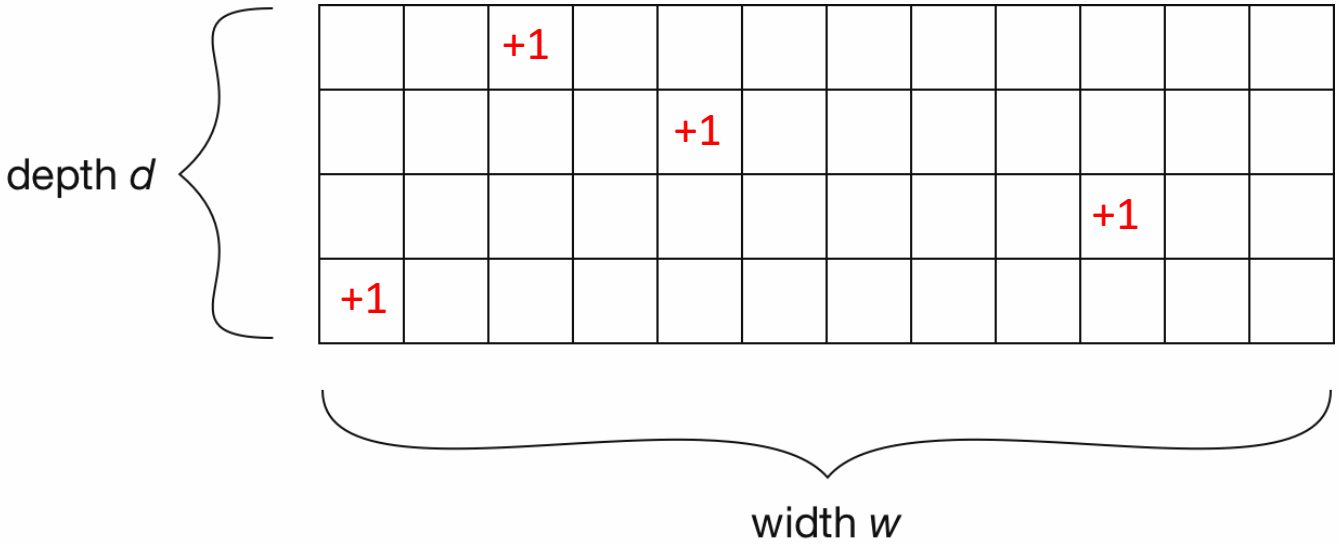

https://en.wikipedia.org/wiki/Bloom_filter

# Count-min sketch

Возьмем d попарно независимых хэш-функции с w корзинами

Для элемента возьмем хэши и увеличим значение в этих корзинах

Но теперь для одного элемента у нас много счетчиков – возьмем минимум




https://florian.github.io/count-min-sketch/

# Count-min sketch

С вероятностью $1 - \delta$ ошибка будет не больше чем $\varepsilon \sum counts$

Выбрать количество корзин и хэшей можно через $\mathrm{w} = \left\lceil \frac{e}{\varepsilon} \right\rceil$, $d = \left\lceil \ln \frac{1}{\delta} \right\rceil$


Поиск похожих
Similarity search

# Что такое?

- Поиск наиболее похожих элементов (документов, товаров, пользователей…) согласно заданной метрике близости или расстояния (Жаккард, Косинус, Евклид,…)

- Классические примеры: поиск ближайших соседей, ранжирование, дедупликация документов

# Метрики

- Расстояние – переводит пару элементов некоторого пространства в вещественное число
- Должно удовлетворять условиям
  - Расстояние неотрицательно
  - Расстояние равно нулю только тогда, когда совпадают элементы
  - Расстояние симметрично
  - Неравенство треугольника

- Примеры: Jaccard, Euclidian, Cosine

# Примеры использований

- Рекомендательные системы
- Чат-боты
- Распознавание лиц

# Метрики

- Jaccard: $d(x,y) = 1 - \frac{|x \cap y|}{|x \cup y|}$
- Euclidean: $d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
- Cosine: $d(x,y) = \arccos \frac{\sum x_i * y_i}{\sqrt{\sum x_i^2 y_i^2}}$

# Min-hash

Позволяет быстро посчитать меру Жаккарда

| $Element$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $a$ | 1 | 0 | 0 | 1 |
| $b$ | 0 | 0 | 1 | 0 |
| $c$ | 0 | 1 | 0 | 1 |
| $d$ | 1 | 0 | 1 | 1 |
| $e$ | 0 | 0 | 1 | 0 |

http://www.mmds.org/

# Min-hash

Переставим строки и найдем имя элемента с первым ненулевым значением

| *Element* | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $b$ | 0 | 0 | 1 | 0 |
| $e$ | 0 | 0 | 1 | 0 |
| $a$ | 1 | 0 | 0 | 1 |
| $d$ | 1 | 0 | 1 | 1 |
| $c$ | 0 | 1 | 0 | 1 |

$$h(S1) = a, h(S2) = b, h(S3) = a$$

http://www.mmds.org/

# Min-hash

Вероятность совпадения в такой перестановке – мера Жаккарда

| $Element$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $b$ | 0 | 0 | 1 | 0 |
| $e$ | 0 | 0 | 1 | 0 |
| $a$ | 1 | 0 | 0 | 1 |
| $d$ | 1 | 0 | 1 | 1 |
| $c$ | 0 | 1 | 0 | 1 |

$$h(S1) = a, h(S2) = b, h(S3) = a$$

http://www.mmds.org/

# Min-hash

- Но переставлять строки – дорого
  - Хэш эквивалентен перестановке
  - Первый ненулевой элемент имеет минимальное значение функции
- Как оценить вероятность?
  - Используем несколько перестановок (несколько разных хэшей) – сигнатура
  - Посчитаем долю совпадений

# Min-hash

Возьмем две хэш-функции: $x \ mod \ 5$ и $3x - 2 \ mod \ 5$

$J \approx 0.2$

| | | *Element* | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|---|---|
| 1 | 1 | $a$ | 1 | 0 | 0 | 1 |
| 2 | 4 | $b$ | 0 | 0 | 1 | 0 |
| 3 | 2 | $c$ | 0 | 1 | 0 | 1 |
| 4 | 0 | $d$ | 1 | 0 | 1 | 1 |
| 0 | 3 | $e$ | 0 | 0 | 1 | 0 |

http://www.mmds.org/

# Approximate Nearest Neighbors

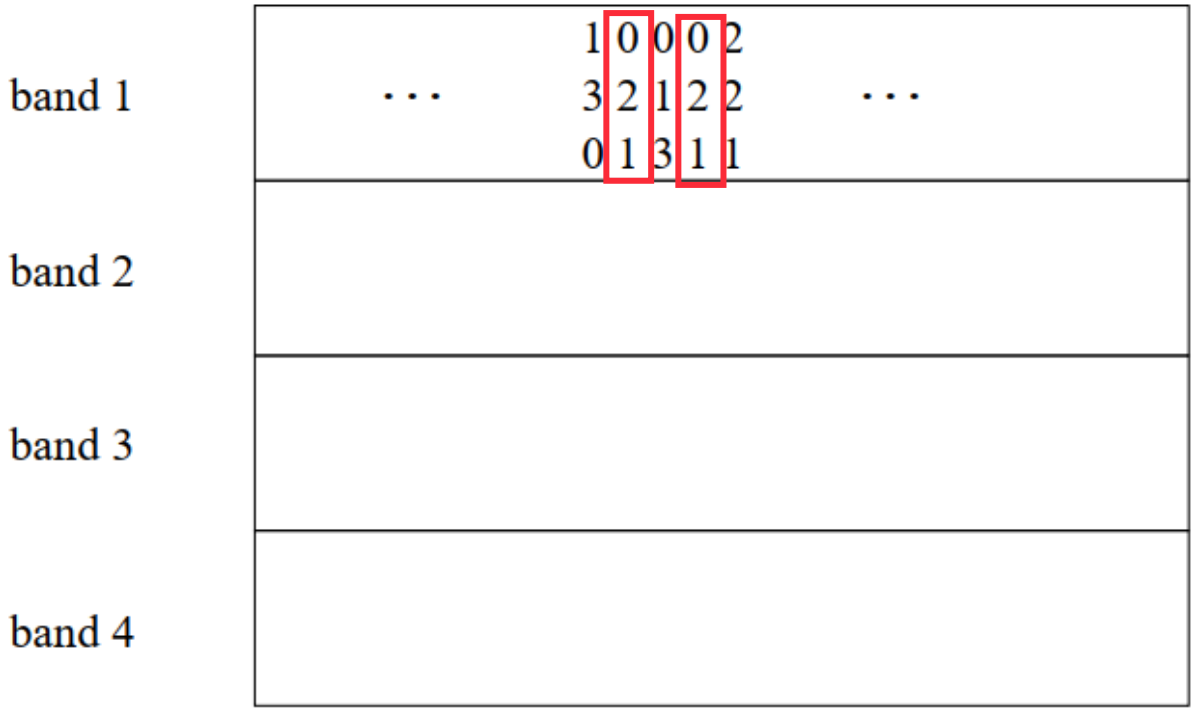
Мы научились быстро сравнивать два документа

Однако даже такой подход не позволит быстро находить ближайшие элементы на входящий запрос

Давайте выберем только потенциально наиболее близких кандидатов и проведем медленное сравнение уже на этом подмножестве

# Locality-sensitive hashing (LSH) for MinHash

Разделим наши MinHash сигнатуры на полоски (bands) и посмотрим на коллизии внутри них. Если коллизии есть, то уже полноценно сравниваем эти документы

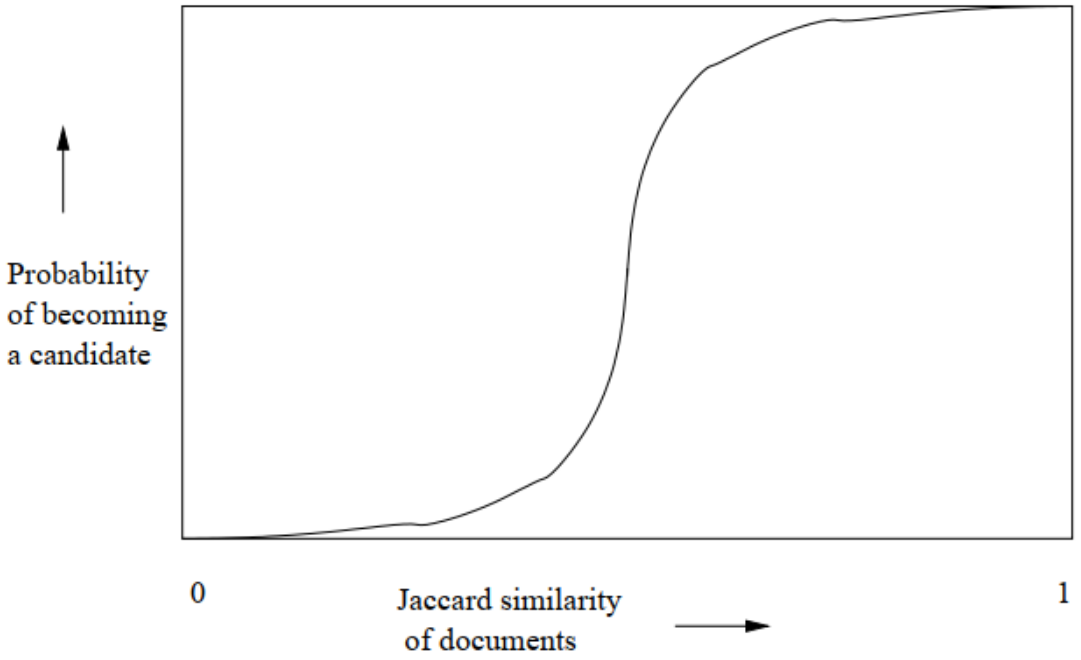



http://www.mmds.org/

# LSH for MinHash

Вероятность стать кандидатом $1 - (1 - s^r)^b$

s – близость Жаккарда, r – ширина полосы, b – количество полос




| $s$ | $1 - (1 - s^r)^b$ |
|---|---|
| .2 | .006 |
| .3 | .047 |
| .4 | .186 |
| .5 | .470 |
| .6 | .802 |
| .7 | .975 |
| .8 | .9996 |

http://www.mmds.org/

# Bucketed random projection (Euclidean)

Проводим прямую

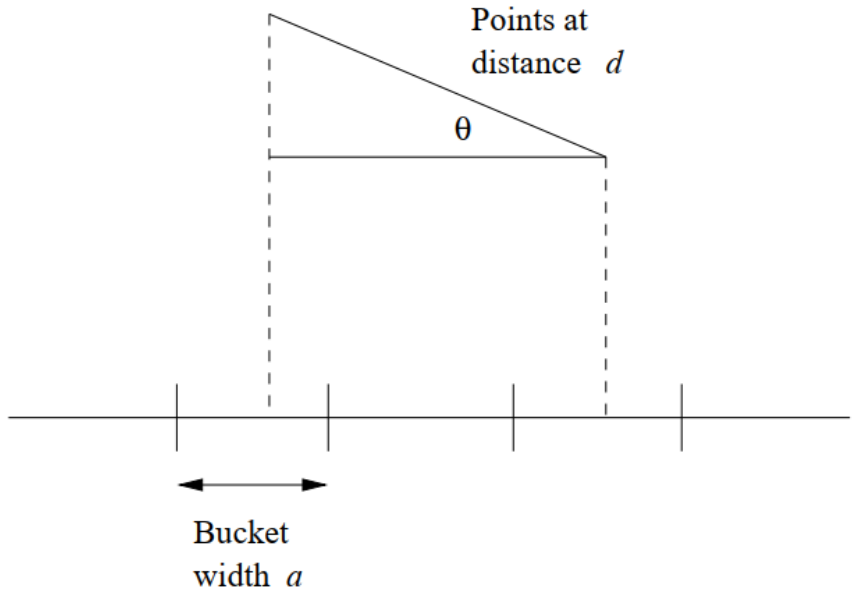
Делим на корзинки равной длины

Смотрим попали ли две точки в одну корзину

Для проекции используем хэш-функцию

$h(x) = \left\lfloor \frac{v \cdot x}{a} \right\rfloor$

$v$ – случайный единичный вектор




http://www.mmds.org/

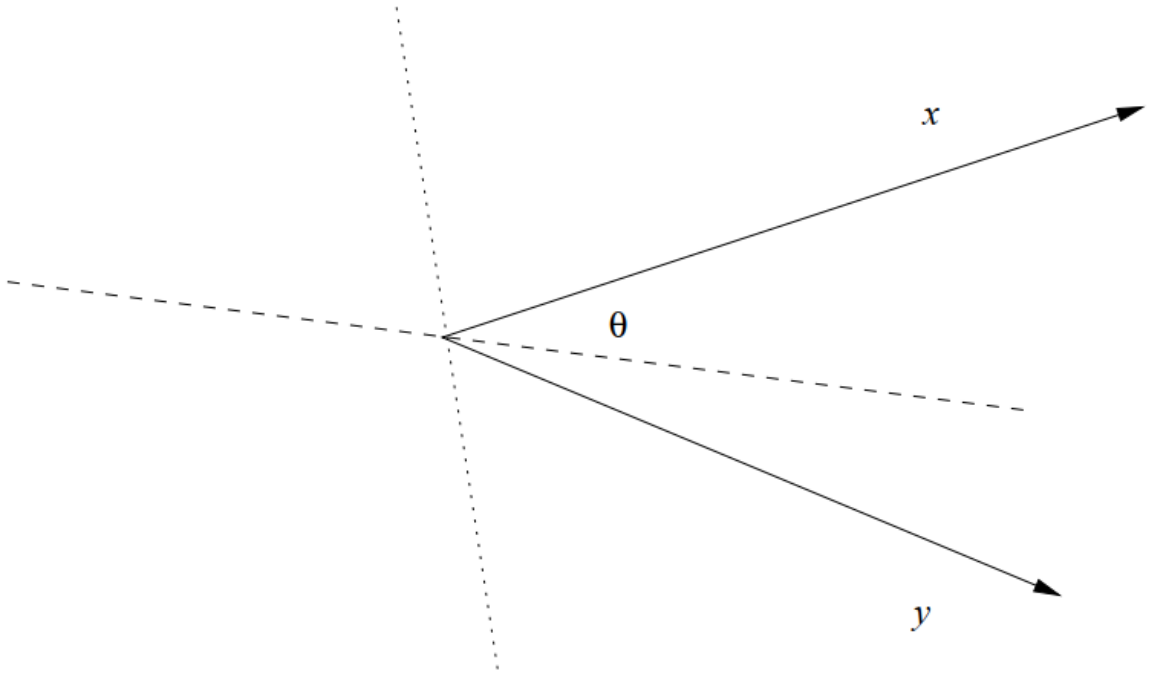
# Random Hyperplanes (Cosine)

Берем случайную плоскость

Перемножаем на нормаль и смотрим на знак

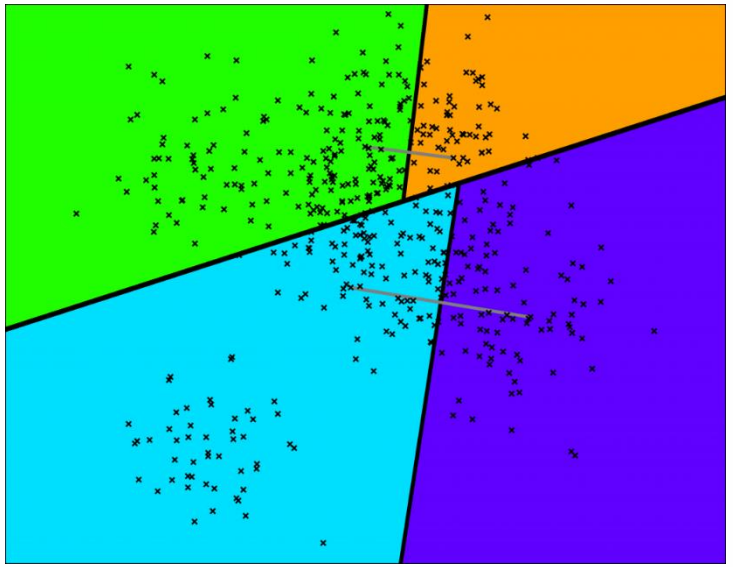
Вероятность того, что находятся с одной стороны равно углу между векторами.

Вместо того, чтобы семплировать случайную плоскость давайте использовать вектор из 1 и -1

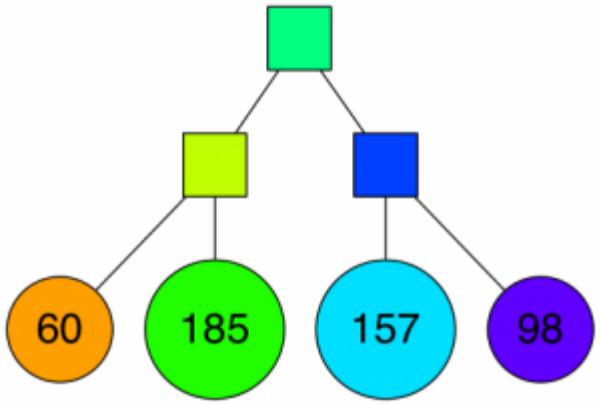



http://www.mmds.org/

# Annoy



- Выберем две случайные точки и проведем к ним нормаль
- Повторим для подплоскости, пока в листе не будет не будет не больше заданного K элементов
- Сохраним индекс в бинарное дерево




- Но одно дерево слишком неточное
- Соберем лес!

https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html

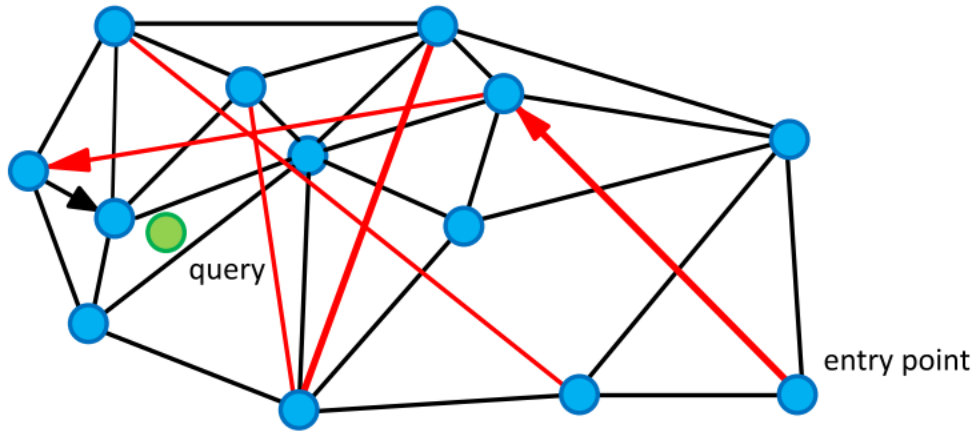
# Navigable Small World (HNSW)

Строим Small World граф по данным

При запросе попадаем на случайную вершину

Идем в ближайшую к запросу соседнюю вершину

Повторяем, пока не окажемся в ближайшей точке.




https://habr.com/ru/company/mailru/blog/338360/

# Hierarchical Navigable Small World (HNSW)

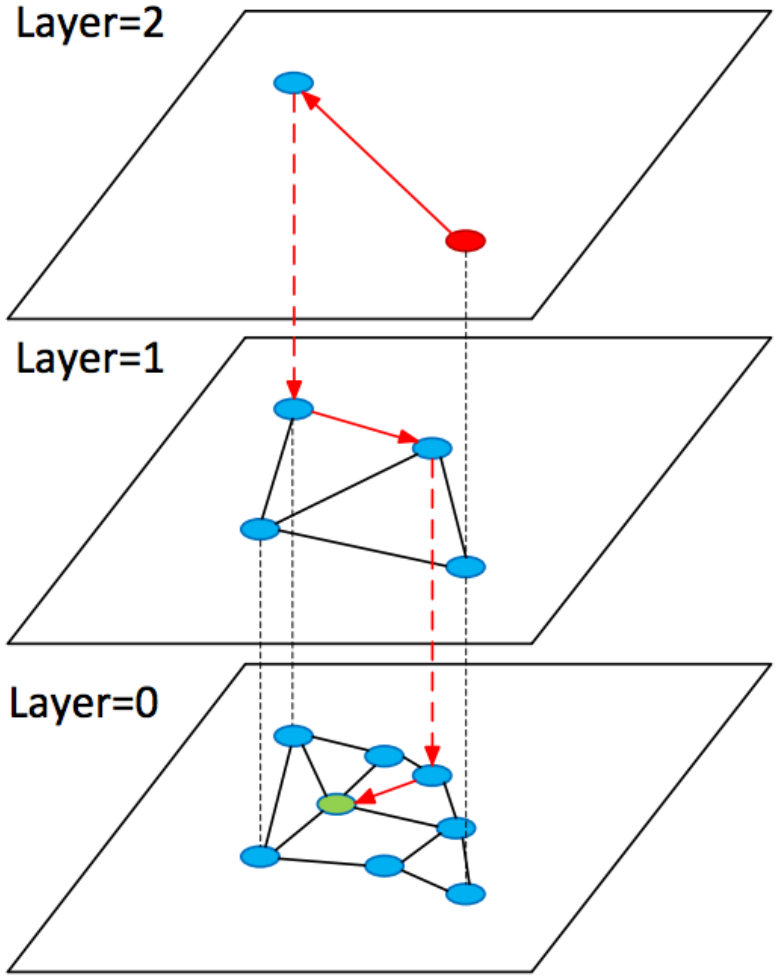
Теперь разделим граф на слои

Все точки со слоя $n$ переходят на слой $n + 1$

Выбираем случайную точку на верхнем слое

Используем механизм NSW

Как только нашли ближайшую точку на слое, спускаемся ниже




https://habr.com/ru/company/mailru/blog/338360/

# Facebook AI research Similarity Search (FAISS)

FAISS состоит из трех основных частей

1. Asymmetric distance computation (ADC)
2. Inverted file (IVF)
    - Для центроида храним сразу список всех векторов
3. Product quantization (PQ)

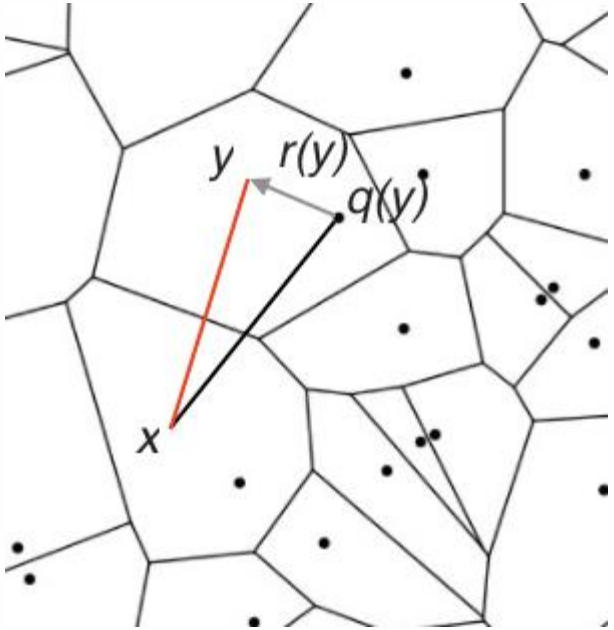
# FAISS: ADC

Воспользуемся идеей корзинок и разделим наше пространство на кластеры с помощью K-Means

Для репрезентации корзины возьмем вектор центрального элемента

При запросе находим ближайший центр

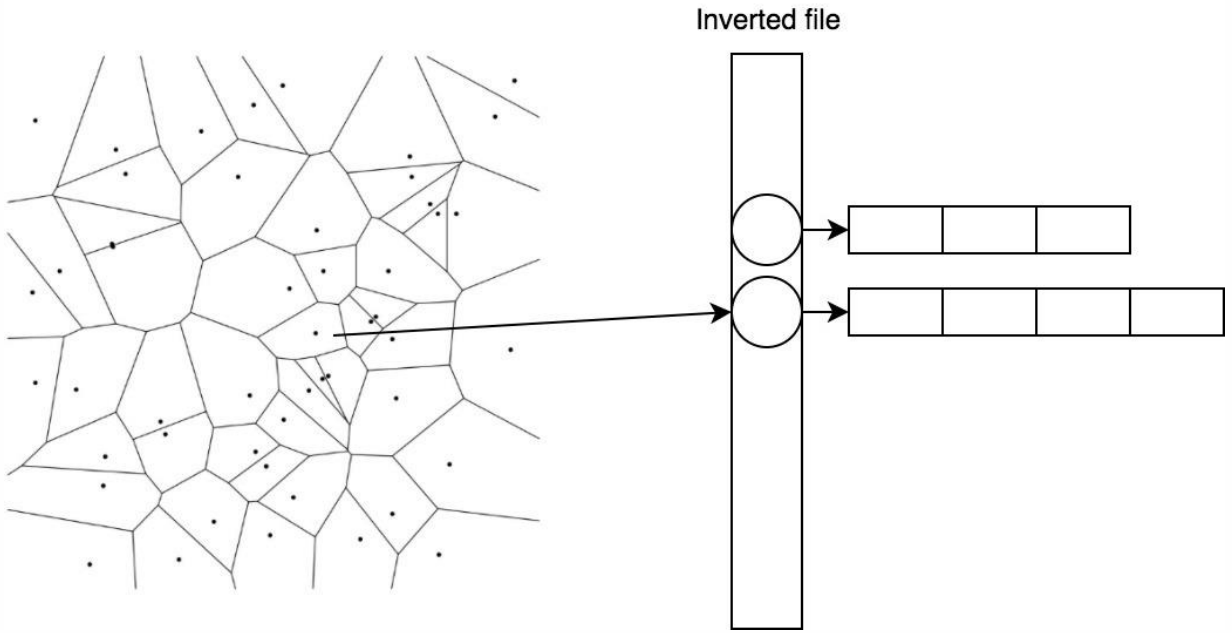
Достаем элементы кластера через IVF




https://habr.com/ru/company/mailru/blog/338360/

# FAISS: IVF

Для центров кластера просто храним список элементов в нем



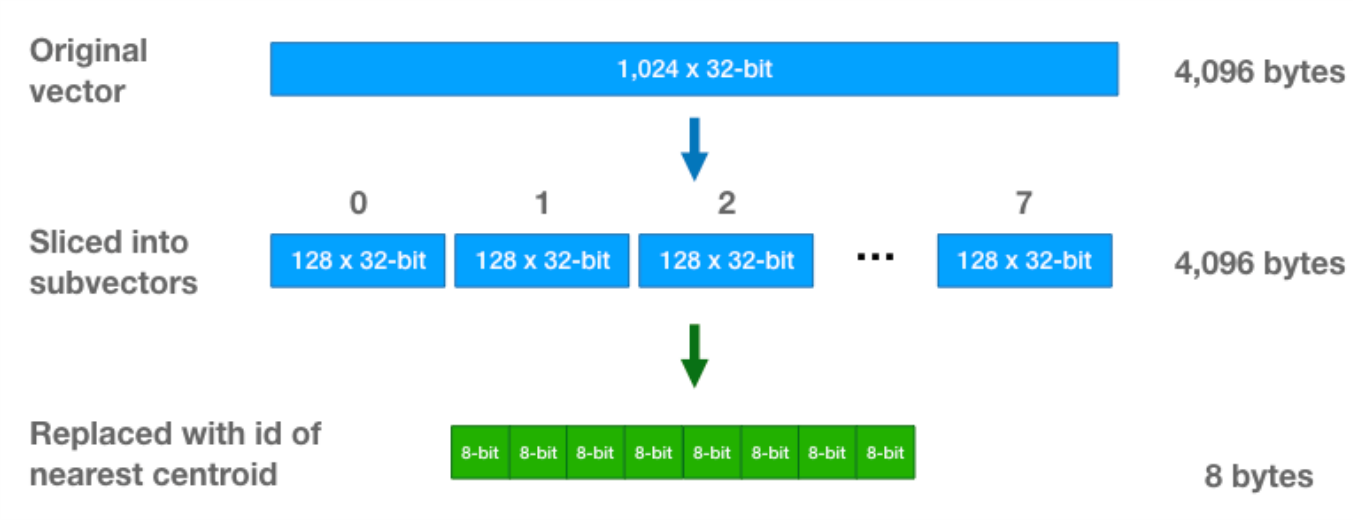

https://habr.com/ru/company/mailru/blog/338360/

# FAISS: PQ

Вычитаем элементы центроида из вектора

Делим полученный вектор на корзинки

Каждую из частей кластеризуем и заменяем индексом центра




https://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

# FAISS: Поиск

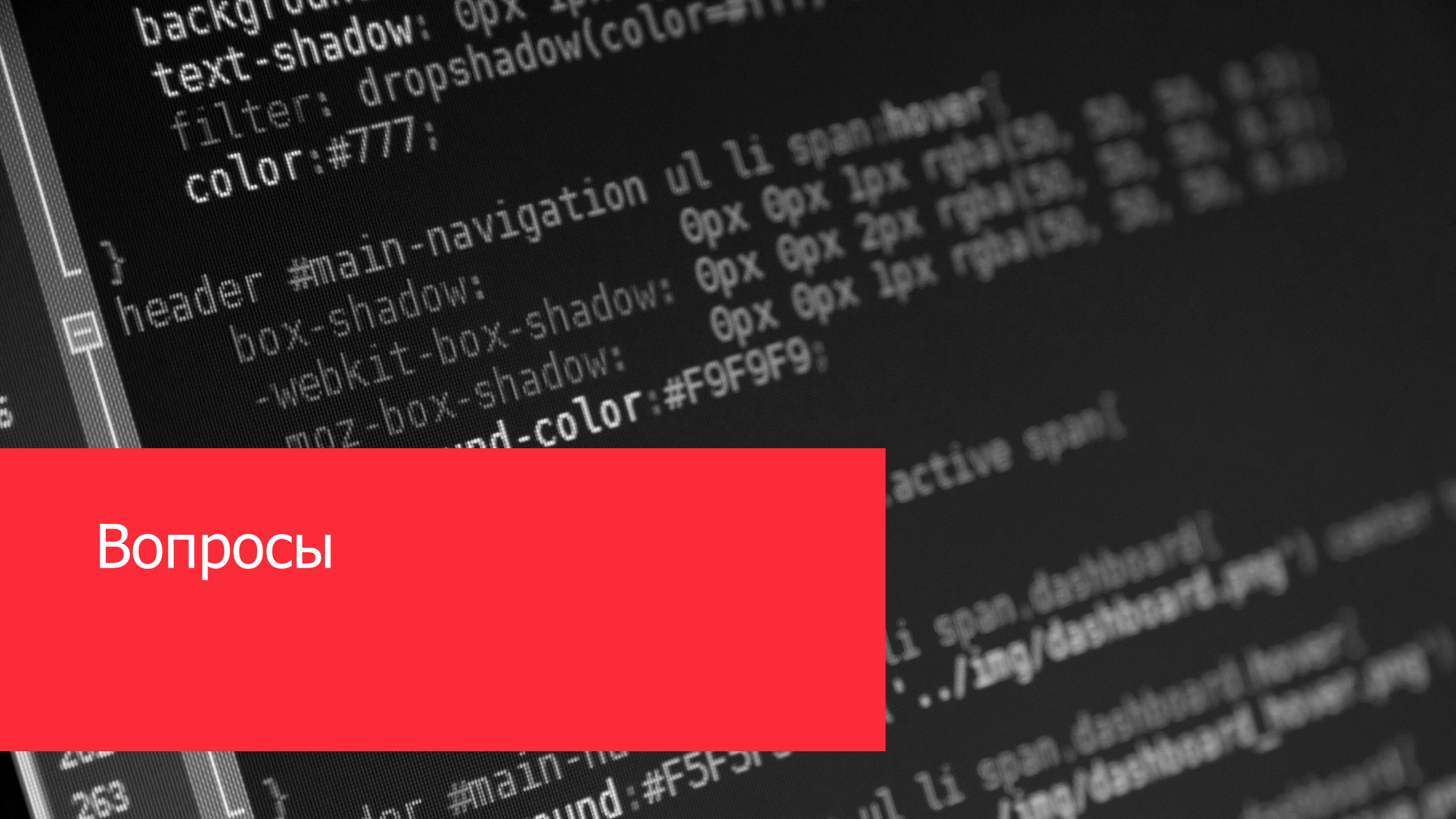
- При запросе находим несколько ближайших центров кластеров
- Достаем элементы кластера через IVF
- Вектор запроса кодируется через PQ
- Расстояние от запроса до элемента определяется, как сумма расстояний от центров кластеров между всеми корзинками
- Достаем ближайших

# Выводы

Хэши позволяют многое ускорить

- Подготовка фичей
- Фильтрация данных
- Подсчет статистик
- Оценка похожести
- Поиск ближайших

Но нужно помнить, что они вносят ошибку


text-shadow: 0px
filter: dropshadow(color=
color:#777;
header #main-navigation ul li span:hover{
box-shadow:
-webkit-box-shadow: 0px 0px 1px
-moz-box-shadow: 0px 0px 2px
0px 0px 1px
-color:#F9F9F9;
active span{

# Вопросы

# Домашнее задание

Реализовать Random Hyperplanes LSH (для косинусного расстояния) в виде Spark Estimator

Основное задание (100 баллов)

1. Посмотреть как устроены MinHashLSH и BucketedRandomProjectionLSH в Spark
2. Унаследоваться от LSHModel и LSH
3. Определить методы createRawLSHModel и keyDistance

Дополнительное задание (30 баллов)

1. Сделать предсказания (на тех же данных и фичах: HashingTf-Idf)
2. Подобрать количество гиперплоскостей и трешхолд по расстоянию