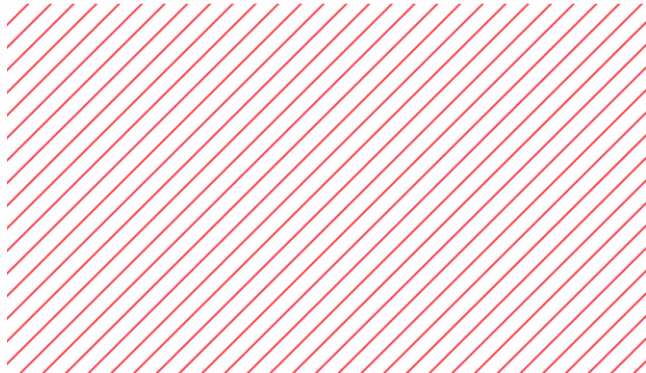
академия больших данных

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 (MurMurHash3 – хороший вариант)
- Выделяем остаток от деления значения хэша на большое простое число
- Для удобства конвертируем в доли – получаем значение CDF равномерного распределения
- Сравниваем CDF с границами вариаций

Bloom filter

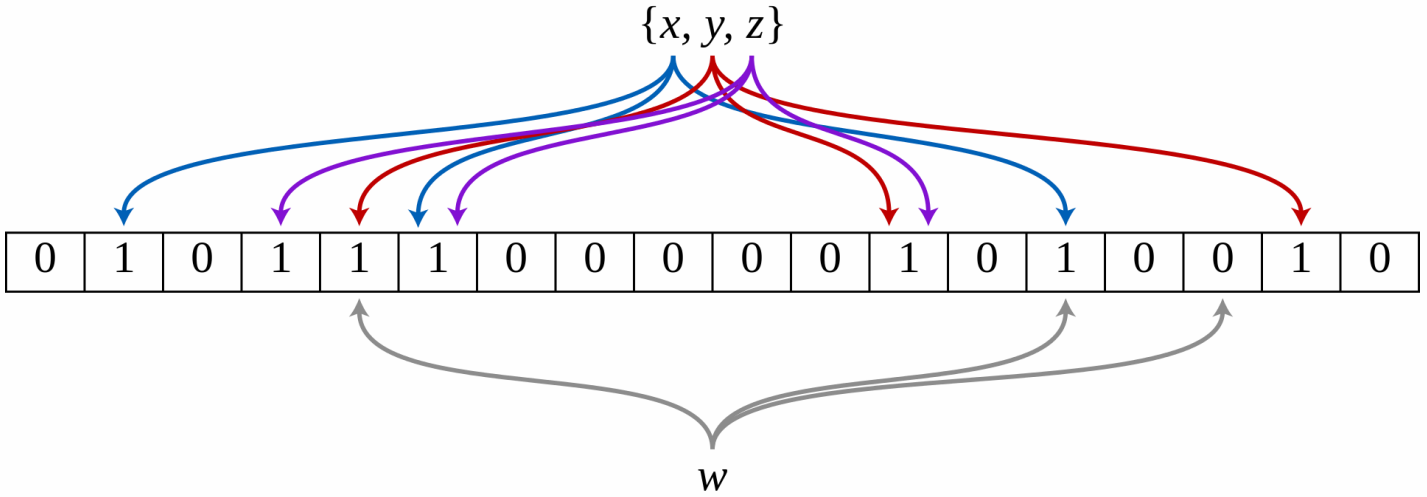
Задача

- У нас есть очень большое множество
- Хотим проверить **входят ли какие-то элементы в него**

Подход в лоб

- Делаем хэш-таблицу по данным
- Смотрим, что хэш для нового элемента уже существует
- Быстро (лукап по хэш-таблице – константа), но требует много места

Bloom filter



Имеет две операции

- добавление в множество
- **проверка на отсутствие**

Состоит из n бит (сначала все 0) и k хэш-функций, которые кладут в один из n бит

Добавление

1. Считаем k хэш-функций от элемента
2. Меняем 0 на 1 (или оставляем 1) в корзинках, которые вернули хэши

Поиск

1. Считаем k хэш-функций от элемента
2. Если какой-то бит равен 0, то элемента нет в множестве

Но могут быть коллизии!

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

Bloom Filter

Но могут быть коллизии!

Давайте посчитаем вероятность False Positive через параметры n, k и количества элементов в множестве (m)

1. Каждая хэш-функция попадает равномерно в одну из n корзинок, тогда вероятность промаха по конкретной корзинке $1 - \frac{1}{n}$
2. Тогда вероятность, что все хэш-функции промахнуться мимо этого бита равна $\left(1 - \frac{1}{n}\right)^k$
3. Всего m элементов, поэтому вероятность, что в заполненном фильтре пустой бит равна $\left(1 - \frac{1}{n}\right)^{km}$
4. Вероятность того, что все хэши выставлены для нового элемента

$$\left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^k \approx \left(1 - e^{-\frac{km}{n}}\right)^k$$

Bloom filter

Вероятность коллизии $p = \left(1 - e^{-\frac{km}{n}}\right)^k$ минимальна при $k = \frac{n}{m} \ln 2$

Подставив k в формулу p получим $\ln p = -\frac{n}{m}(\ln 2)^2$

Отсюда

$$n = -\frac{m \ln p}{(\ln 2)^2} \approx -1.44\, m \log_2 p$$

$$k = -\log_2 p$$

Если у нас $m = 10^6$ элементов и хотим ошибаться не чаще $p = 0.01$

Количество требуемых бит $n \approx 10^7, k \approx 7$

Обычная хэш-таблица при хэше в 32 бита – $3.2 * 10^7$

Count-min sketch

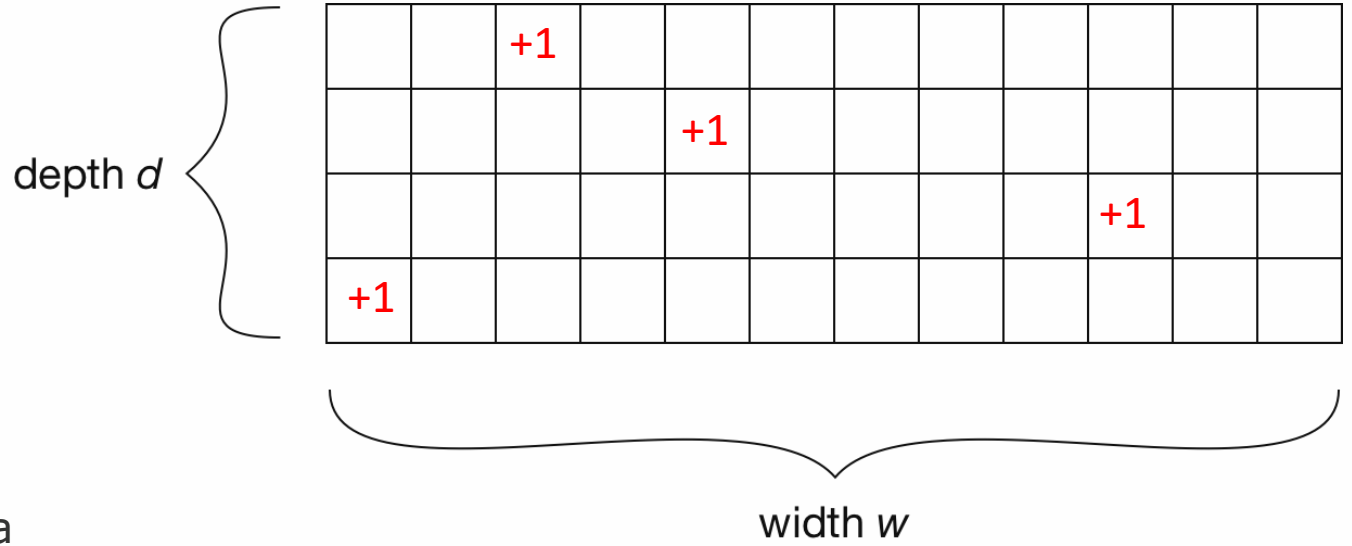
Задача

- Имеем нагруженный поток данных
- Хотим посчитать как часто конкретный элемент встречается в потоке

Подход в лоб

- Делаем хэш-таблицу элемент – встречаемость
- Смотрим по хэшу встречаемость элемента
- Быстро (лукап по хэш-таблице – константа), но требует много места

Count-min sketch



Операции

- Инкремент счетчика для элемента
- Извлечение счетчика для элемента

Состоит из d разных хэш-функций с w корзинками

Добавление

1. Считаем d хэш-функций от элемента
2. Добавляем 1 в соответствующие ячейки

Извлечение

1. Считаем d хэш-функций от элемента
2. Берем минимум из полученных значений в ячейках

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: k-ближайших соседей, ранжирование (рексистемы,чат-боты, распознавание лиц,), 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

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

$$h(S1) = a, h(S2) = c, h(S3) = b, h(S4) = a$$

http://www.mmds.org/

Min-hash

Вероятность совпадения в такой перестановке – мера Жаккарда

Всего возможно три варианта событий

1. Оба элемента в строке равны 1 (количество элементов в пересечении)
2. Только один из элементов в строке имеет 1 (объединение – пересечение)
3. В обоих строках нули

В данном случае нам интересны только 1 и 2.

Вероятность того, что 1 будет раньше 2 равна $\frac{|1|}{|1|+|2|}$, что совпадает с близостью Жаккарда

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 три хэш-функции: $x \ mod \ 5$, $x^3 - 1 \ mod \ 5$ и $3x - 2 \ mod \ 5$

$$J = \frac{2}{3}$$

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

http://www.mmds.org/

Approximate Nearest Neighbors

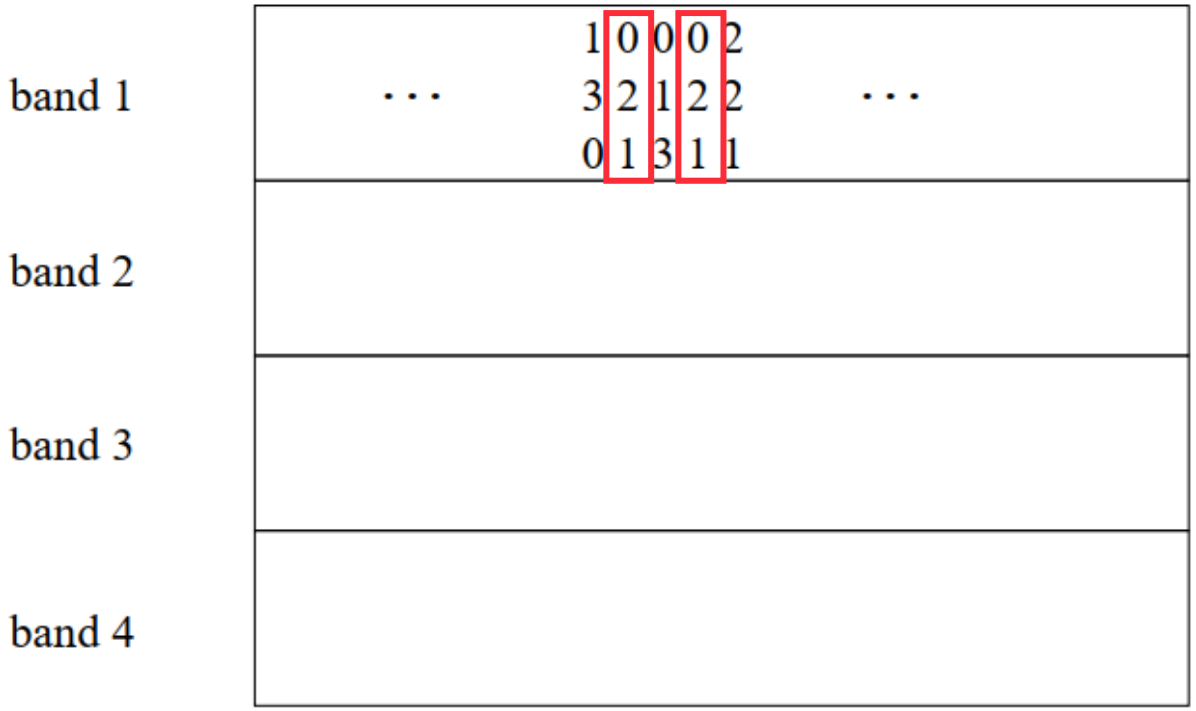
Мы научились быстро сравнивать два документа

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

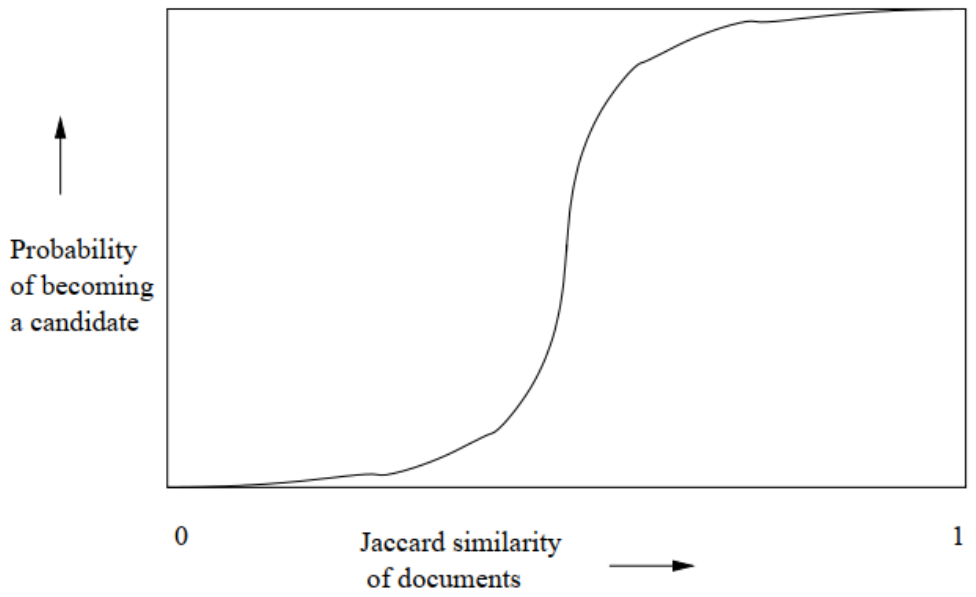


http://www.mmds.org/

LSH for MinHash

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

1. Пусть близость между двумя документами равна s
2. Тогда вероятность, что сигнатуры совпадут в одной полоске – s^r
3. Вероятность несовпасть хотя бы в одной строке полоски равна $1 - s^r$
4. Вероятность, что одна строка не совпадет везде равна $(1 - s^r)^b$
5. Тогда вероятность, что кандидаты полностью совпадут хотя бы в одной полосе равна $1 - (1 - s^r)^b$



http://www.mmds.org/

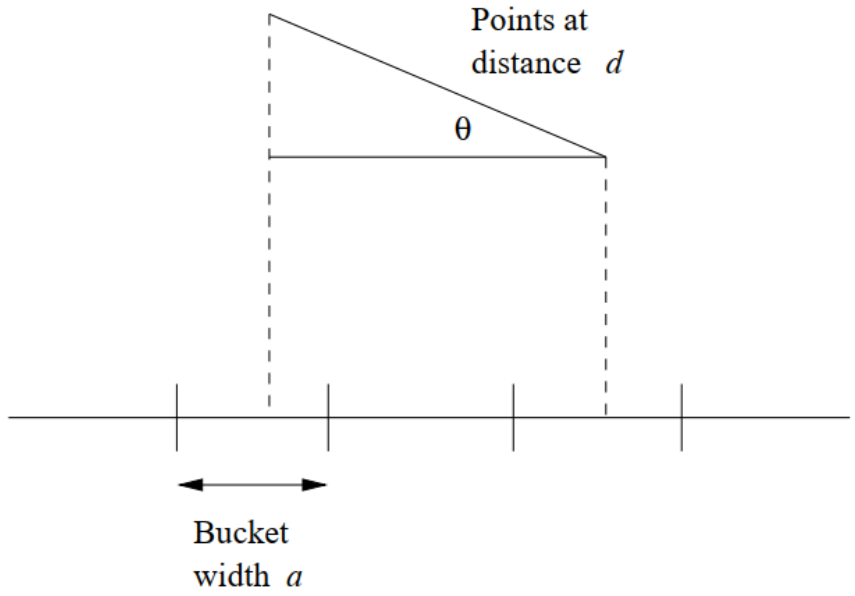
Bucketed random projection (Euclidean)

1. Проводим прямую
2. Делим на корзинки равной длины
3. 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

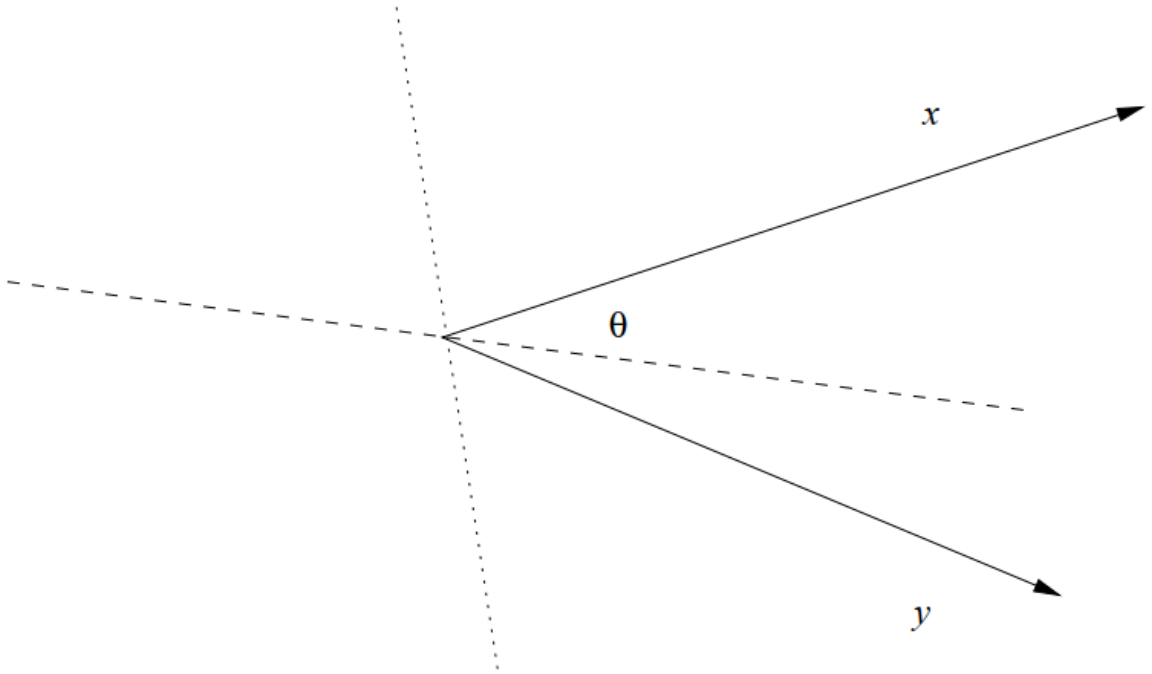


http://www.mmds.org/

Random Hyperplanes (Cosine)

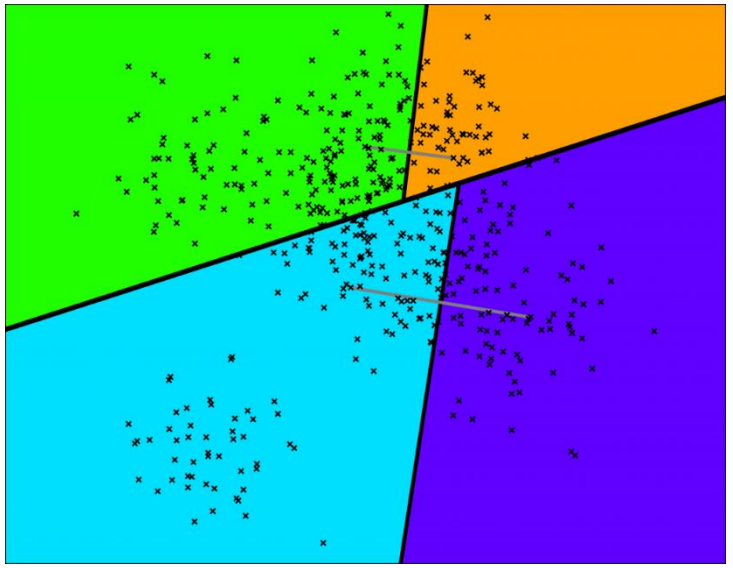
1. Берем случайную плоскость
2. Перемножаем вектора на нормаль и смотрим на знаки
3. Вероятность их расположения по одной стороне равнп углу между векторами.

Вместо того, чтобы семплировать случайнук
вектор из 1 и -1

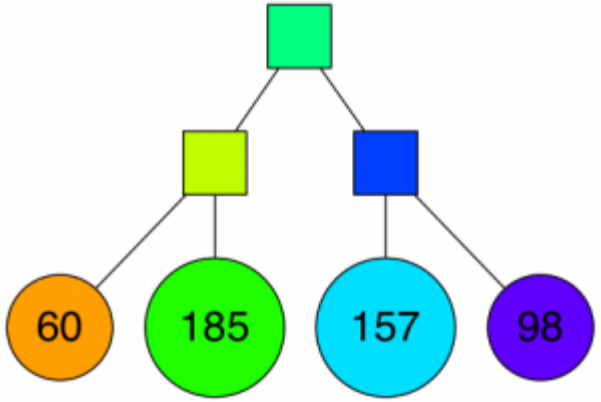


http://www.mmds.org/

Annoy



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

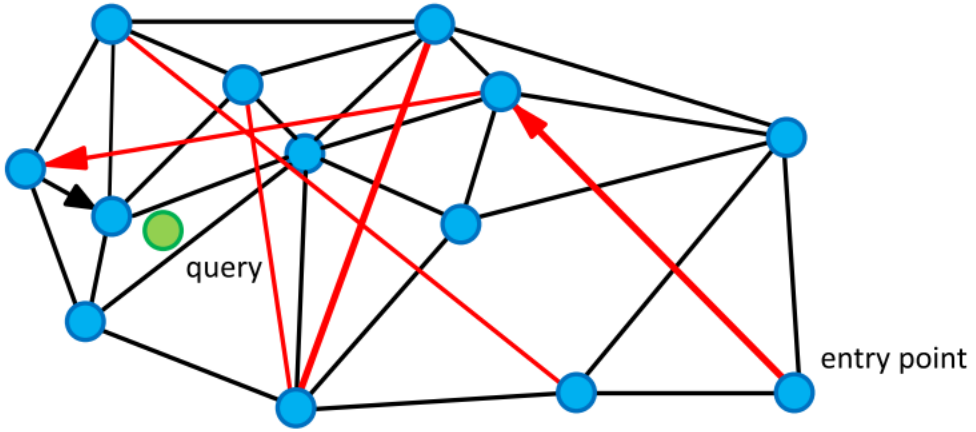


- Но одно дерево слишком неточное
- Соберем лес!

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

Navigable Small World (NSW)

1. Строим Small World граф по данным
2. При запросе попадаем на случайную вершину
3. Идем в ближайшую к запросу соседнюю вершину
4. Повторяем, пока не окажемся в ближайшей точке.



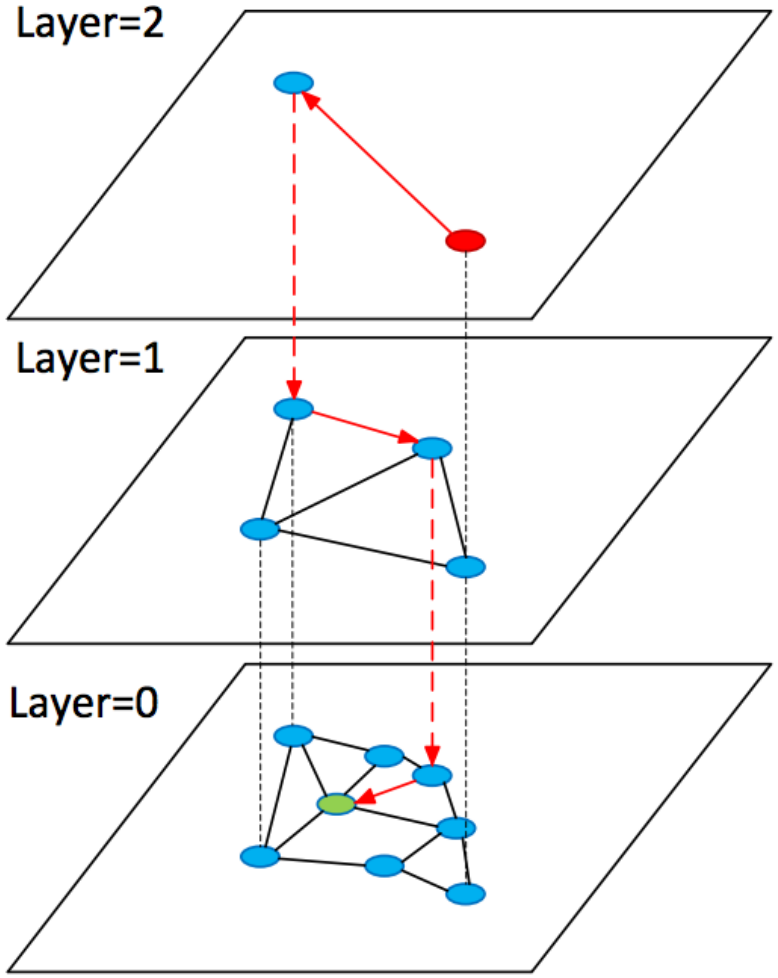
https://habr.com/ru/company/mailru/blog/338360/

Hierarchical Navigable Small World (HNSW)

Теперь разделим граф на слои

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

1. Выбираем случайную точку на верхнем слое
2. Используем механизм NSW
3. Как только нашли ближайшую точку на слое, спускаемся ниже



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

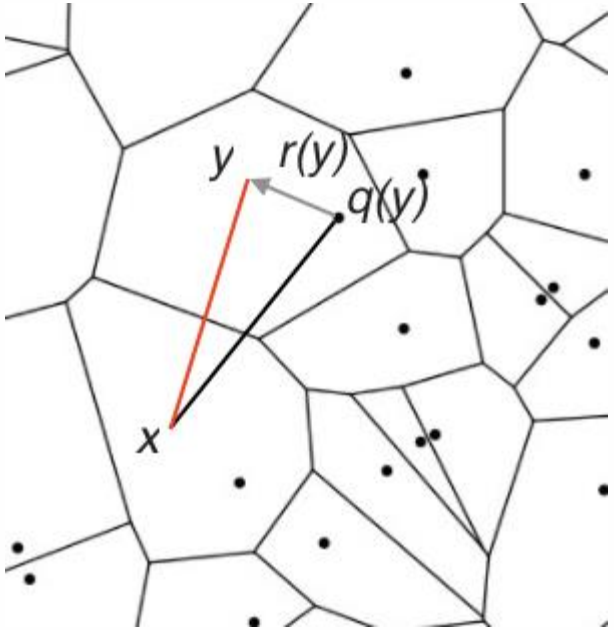
Facebook AI research Similarity Search (FAISS)

FAISS состоит из трех основных частей

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

FAISS: ADC

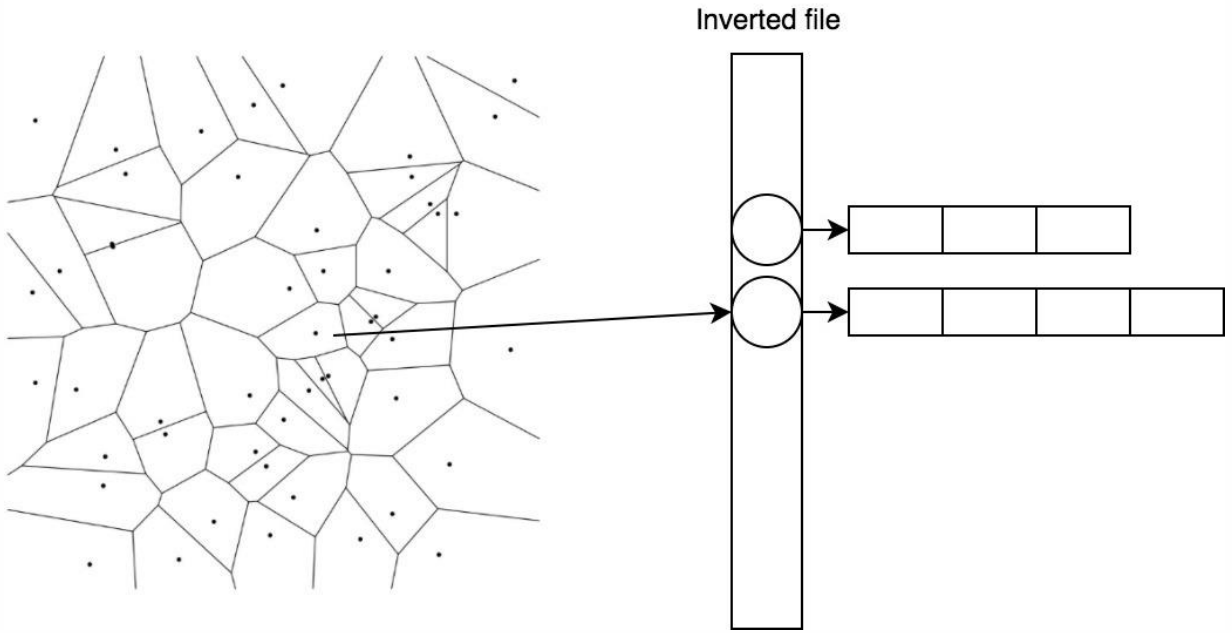
- Воспользуемся идеей корзинок и разделим наше пространство на кластеры с помощью K-Means
- Для репрезентации корзины возьмем вектор центрального элемента
- При запросе находим ближайший центр
- Достаем элементы кластера через IVF



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

FAISS: IVF

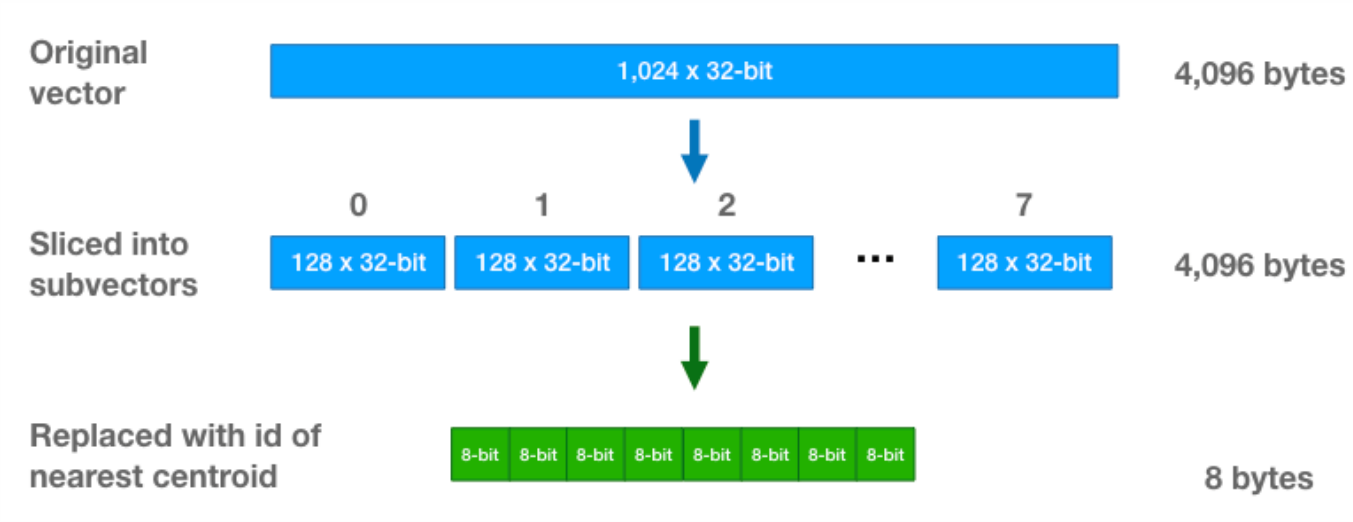
Для центров кластера просто храним список элементов в нем



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

FAISS: PQ

1. Вычитаем элементы центроида из вектора
2. Делим полученный вектор на корзинки
3. 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
- Остаток вектора от запроса до центра кластера 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

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 недостающие методы для модели (hashFunction, hashDistance keyDistance, write) и для LSH (createRawLSHModel, copy, transformSchema)

Дополнительное задание (30 баллов)

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