

Алгоритмы в индустрии

29/04/2023

Компоненты системы

- Интерфейс для пользователя
- "Бизнес"-логика, специфична для продукта
- Основные модули системы, на основе которых создается функционал

Будем обозначать за UI, Logic, Core

Пример: Subway Surfers

- UI: мобильное приложение, свайпы для смены путей.
- Logic: после второго столкновения с препятствием тебя ловит контролер.
- Core: проверка на столкновения.

Пример: Гугл-календарь

- UI: веб-сайт.
- Logic: напомнить за 15 и 5 минут до события.
- Core: база данных для сохранения событий, отправка уведомлений.

Пример: Телеграм-бот для скачивания записей лекций

- UI: Телеграм + Markdown для верстки сообщений.
- Logic: Отправка актуальной ссылки и скачанного видео в нужном разрешении на скорости x1.5.
- Core: Выгрузка видео с youtube.

Core: Алгоритмы

Алгоритмы не обязаны лежать в Core. Но, как правило, если вы пользуетесь алгоритмом, он является core-модулем.

Скорее всего, это означает, что какой-то программист занимается конкретно этим core-модулем, а вы просто пользуетесь его проектом

Организация

- FaaS (например, REST API)
- Third-party libraries (например, open source)
- Самописные модули

Пример: OpenAI API

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.Completion.create(
    model="text-davinci-003",
    prompt="Say this is a test",
    max_tokens=7,
    temperature=0
)
```

```
{
    // ...
    "text": "\n\nThis is indeed a test",
    // ...
}
```

Пример: Boost (на букву А)

- Accumulators
- Algorithm
- Align
- Any
- Array
- Asio
- Assert
- Assign
- Atomic



The Boost Algorithm Library

Marshall Clow

Copyright © 2010-2012 Marshall Clow

Distributed under the Boost Software License, Version 1.0. (See accompanying file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

Table of Contents

Description and Rationale
Searching Algorithms
Boyer-Moore Search
Boyer-Moore-Horspool Search
Knuth-Morris-Pratt Search
C++11 Algorithms
 all_of
 any_of
 none_of
 one_of
 is_sorted
 is_partitioned
 is_permutation
 partition_point
 partition_copy
 copy_if
 copy_n
 iota
C++14 Algorithms
 equal
 mismatch
C++17 Algorithms
 for_each_n
 transform_inclusive_scan
 transform_exclusive_scan
Variations on Copy
 copy_until
 copy_while
 copy_if_until
 copy_if_while
Other Algorithms
 none_of_equal
 one_of_equal
 is_decreasing
 is_increasing
 is_strictly_decreasing
 is_strictly_increasing
 clamp
 clamp_range
 find_not

API

API является описанием интерфейса модуля. Можно считать, что модуль является черным ящиком, который соответствует API.

Предполагается, что при создании Core-компонент их API является API общего назначения и не принимает ничего, что бы относилось к Logic-компонентам.

Callback в API

При тесном взаимодействии Logic и Core можно предоставить возможность исполнения произвольного кода в виде function-as-an-argument.

```
let numbers = vec![1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let even_numbers: Vec<i32> = numbers
    .iter()
    .filter(|&n| n % 2 == 0)
    .map(|&n| n * 2)
    .collect();

println!("Doubled even numbers: {:?}", even_numbers);
```

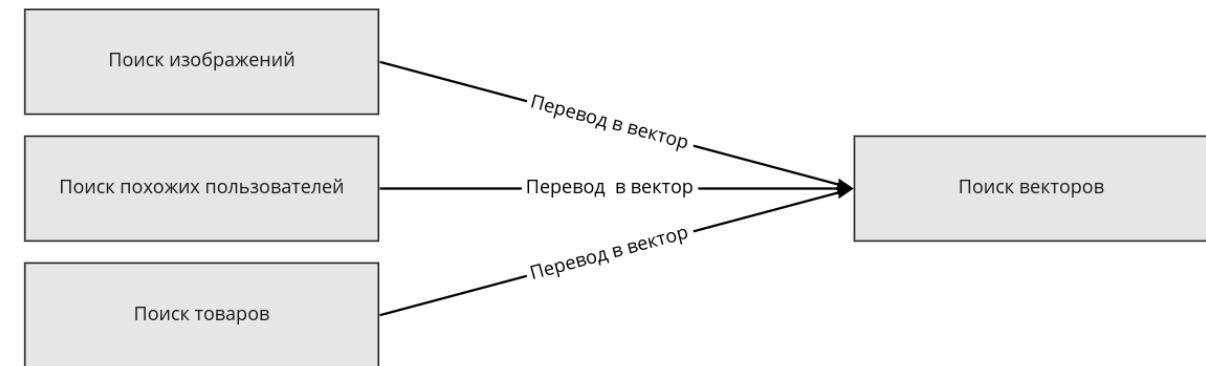
Работа с алгоритмами

- Чтение статей
- Изучение benchmark-ов
- Чтение open-source реализаций
- Интеграция core-модулей

Пример: поисковый сервис

Цель: сделать поисковые сервисы для кастомного типа объектов

Подход: сведение задачи к поиску векторов

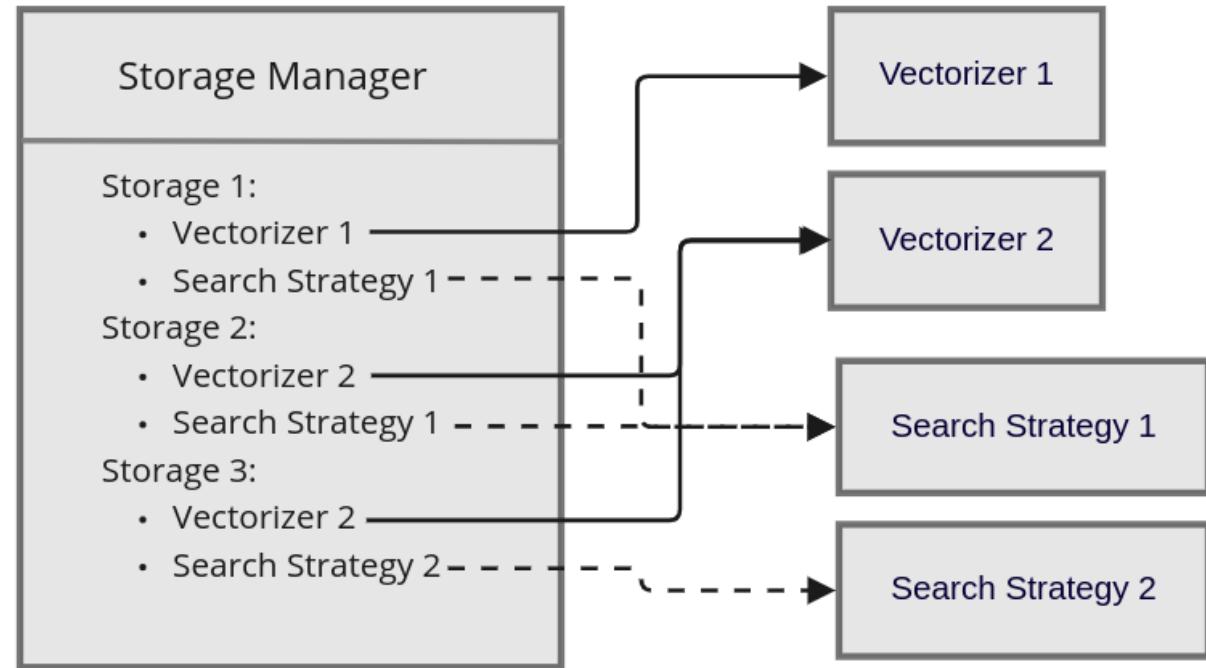


Vectorizer + Search strategy

Для того, чтобы решать поисковую задачу, можно сделать Storage с API:

- Add
- Delete
- Find N

Функционал можно разбить на две Core-компоненты, которые можно комбинировать в произвольных конфигурациях.



API

```
class Vectoriser:  
    def transform(object: SearchObject) -> Vector: ...  
  
class SearchStrategy:  
    def search(vector: Vector) -> List[SearchObject]: ...  
    def add(vector: Vector): ...  
    def remove(vector: Vector): ...
```

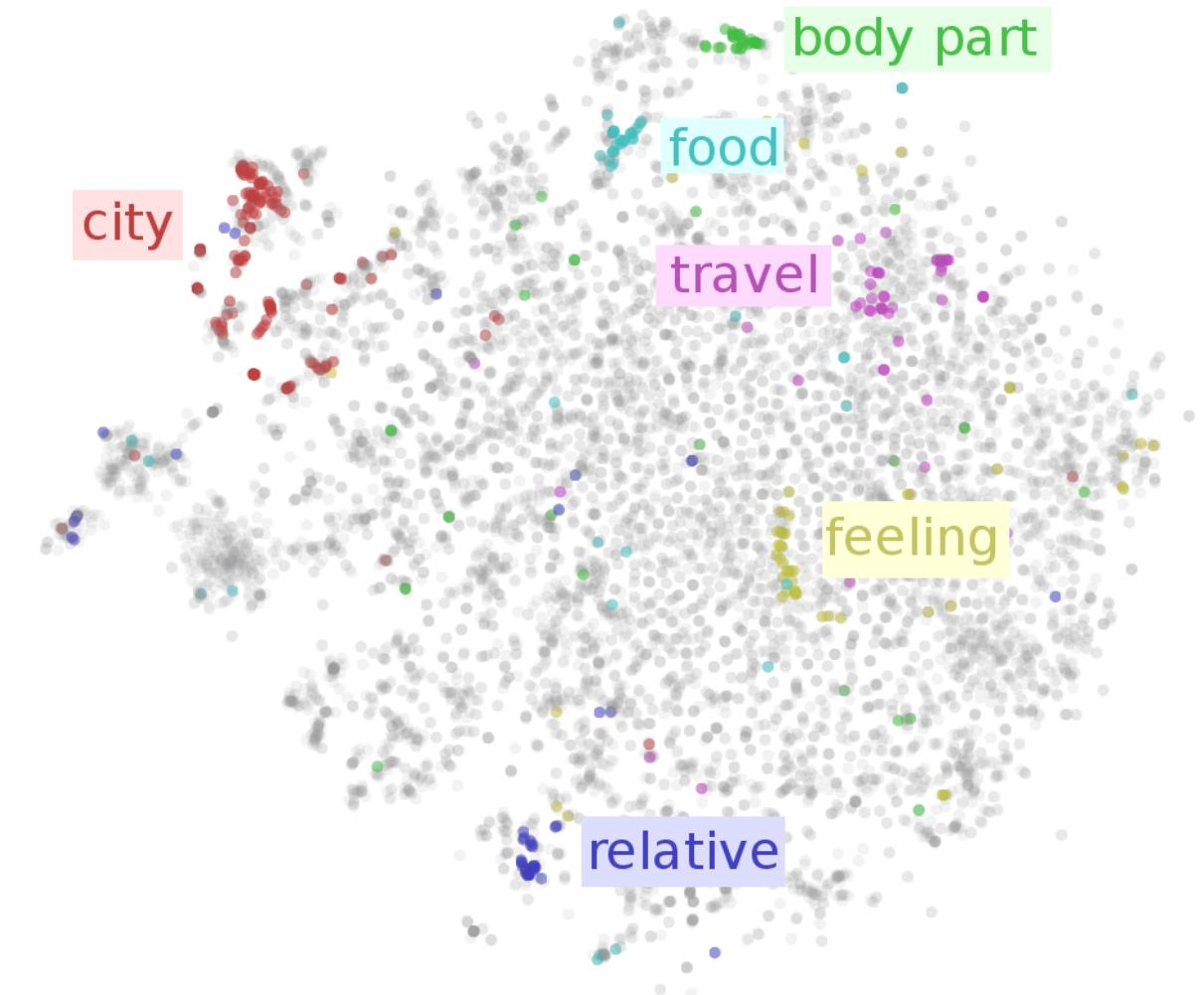
Векторизатор

Нейронные сети можно использовать для создания embedding-ов: вложений объектов в многомерное векторное пространство. При этом похожие объекты соответствуют близким векторам

Стандартная картинка: $1280 \times 720 \times 3$ чисел

Пожатый эмбеддинг: 512 чисел

Эмбеддинг для слов: Word2Vec



Как найти свой эмбеддинг?

Исследование!

Для картинок, например, нормально работает посчитать последний слой сети-классификатора, типа ResNet.

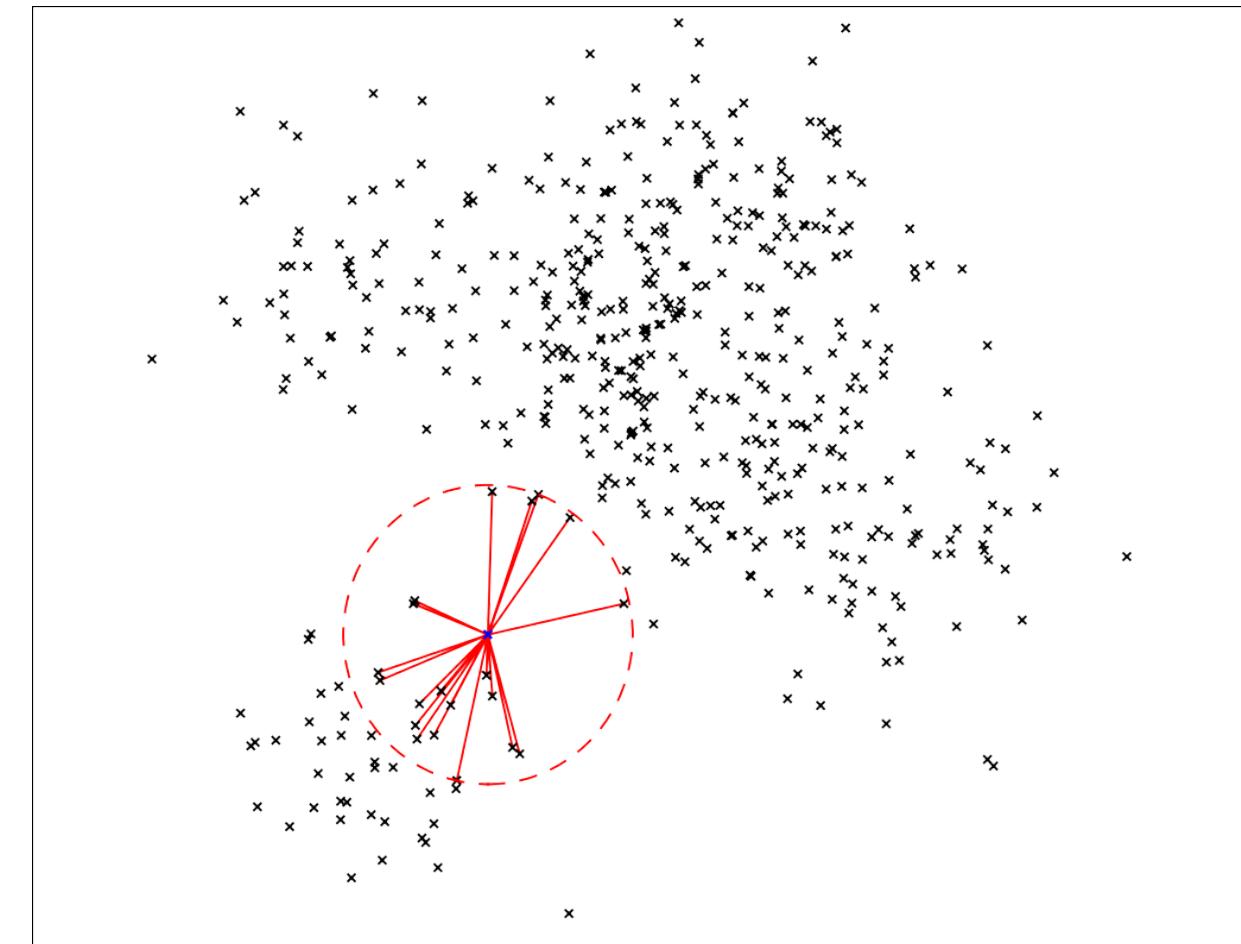
Векторизатор: реализация

Позволить добавлять кастомный python-скрипт, который работает с моделью, получает на вход объект, и выдает вектор.

Поисковая стратегия

Исследуем задачу поиска
похожих векторов.

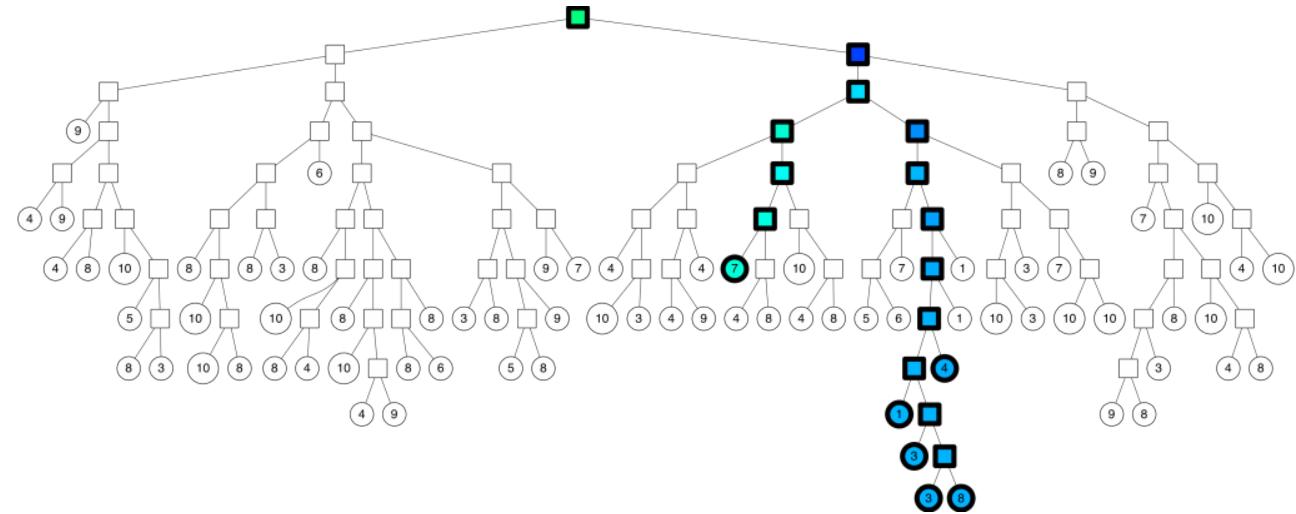
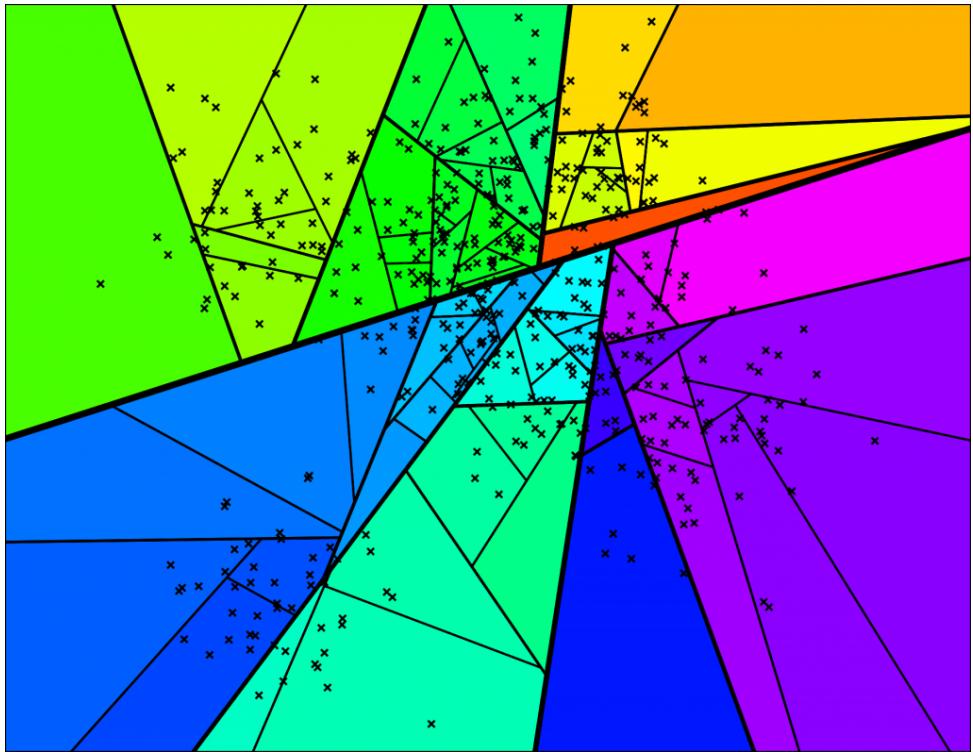
Оказывается, что это отдельный
класс задач ANN (approximate
nearest neighbor).



Решение задачи ANN

Как минимум 4 разных класса решений!

KD-деревья



Spotify

Библиотека annoy

```
from annoy import AnnoyIndex
import random

f = 40 # Length of item vector that will be indexed

t = AnnoyIndex(f, 'angular')
for i in range(1000):
    v = [random.gauss(0, 1) for z in range(f)]
    t.add_item(i, v)

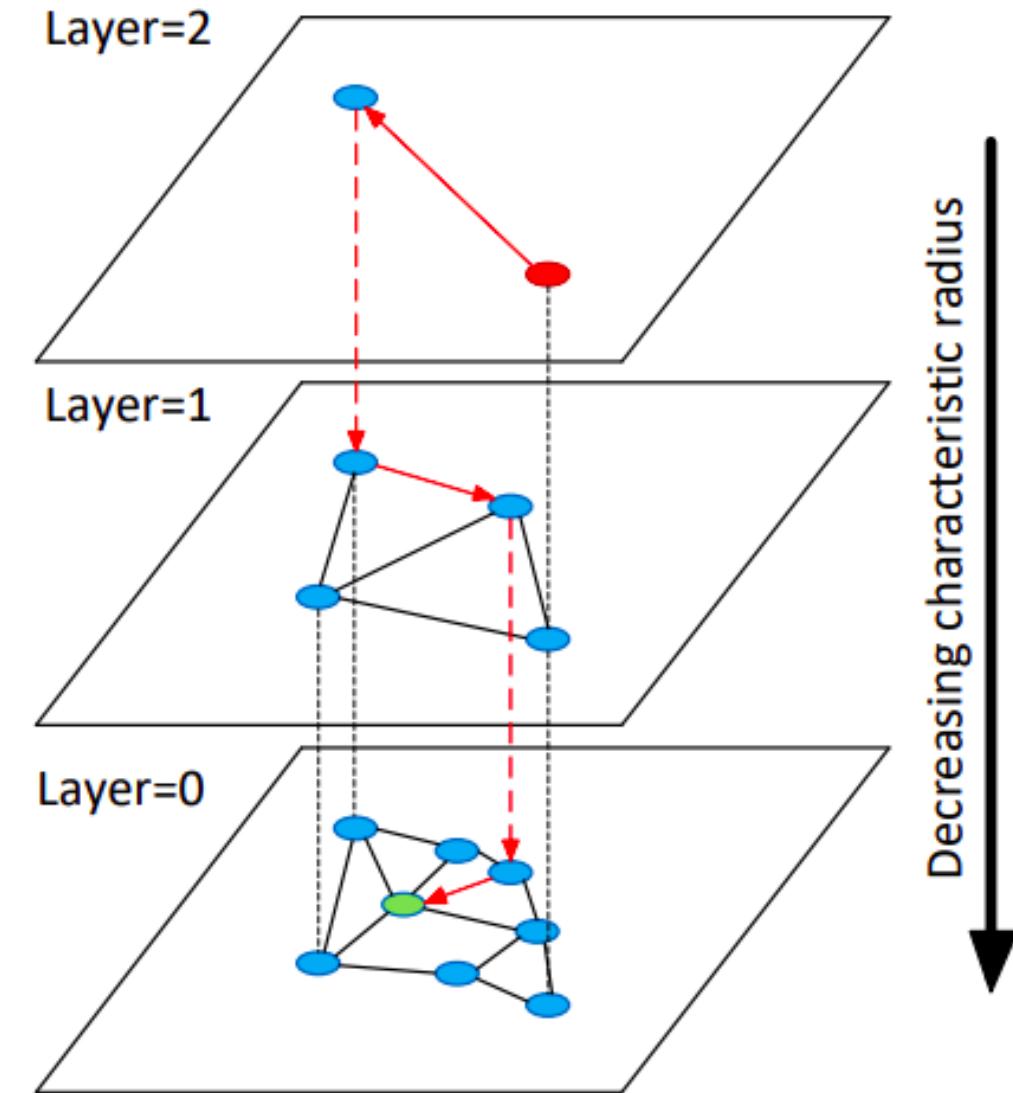
t.build(10) # 10 trees
t.save('test.ann')

# ...

u = AnnoyIndex(f, 'angular')
u.load('test.ann') # super fast, will just mmap the file
v = [random.gauss(0, 1) for z in range(f)]
print(u.get_nns_by_vector(v, 10)) # will find the 10 nearest neighbors
```

HNSW

Комбинация графов и skip-list-ов.



Faiss

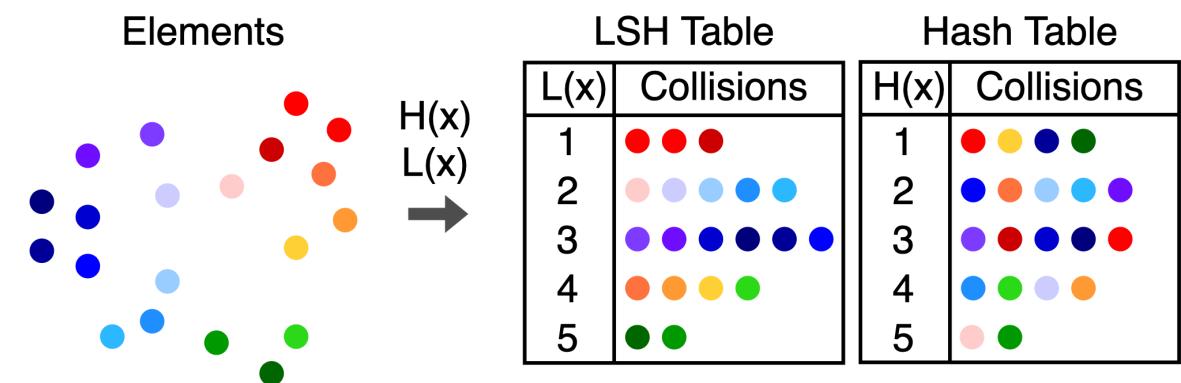
Facebook ANN Library. Есть HNSW и гораздо больше!

```
import numpy as np
import faiss
d = 64                      # dimension
nb = 100000                   # database size
index = faiss.IndexHNSWFlat(train_data)    # build the index
index.add(train)                # add vectors to the index
print(index.ntotal)
k = 4                          # we want to see 4 nearest neighbors
D, I = index.search(test_data, k)
print(I)
print(D)
```

LSH

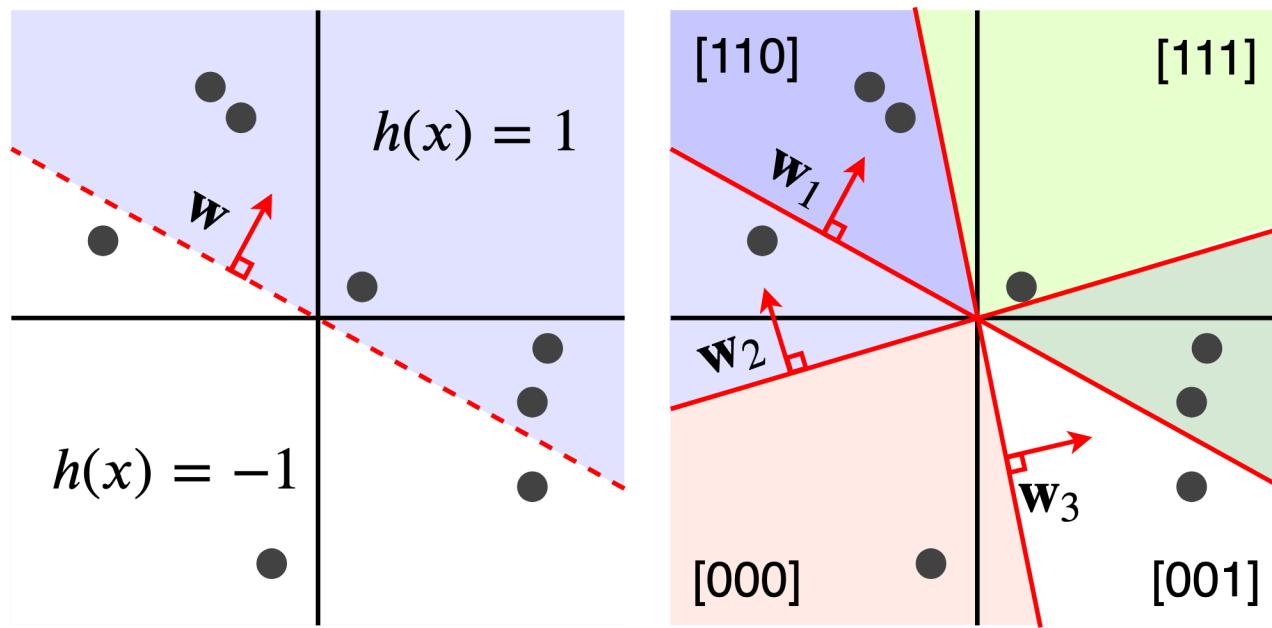
Хитрые хеш-функции, которые хешируют вектора с сохранением расстояния.

Коллизии соответствуют похожим элементам. Функция подбирается под каждое расстояние отдельно.



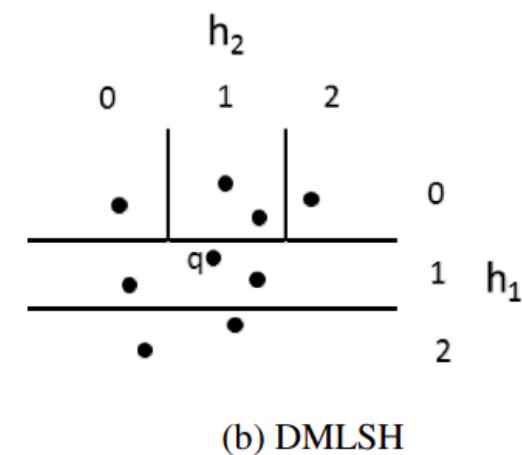
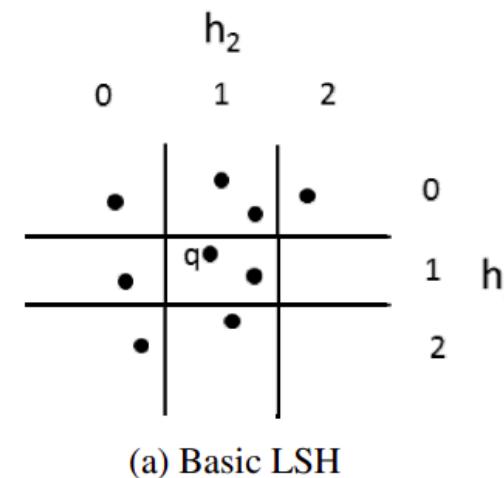
Косинусное расстояние

Генерация случайных прямых, которые делят пространство на "секторы". Хеш от точки соответствует сектору.



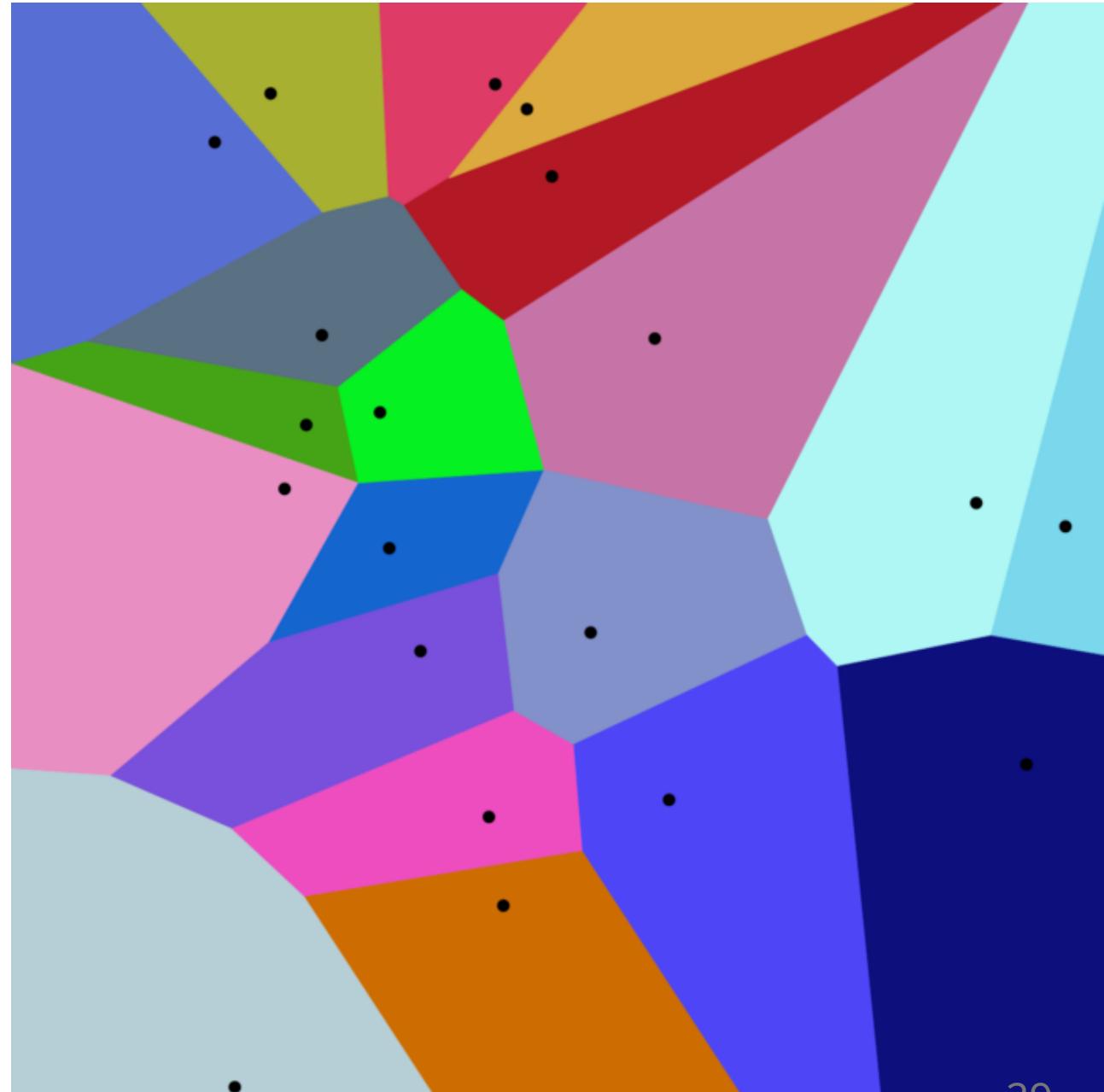
Динамические индексы

Чтобы не создавать сразу все секции хеша, мы можем увеличивать его точность в древовидной структуре

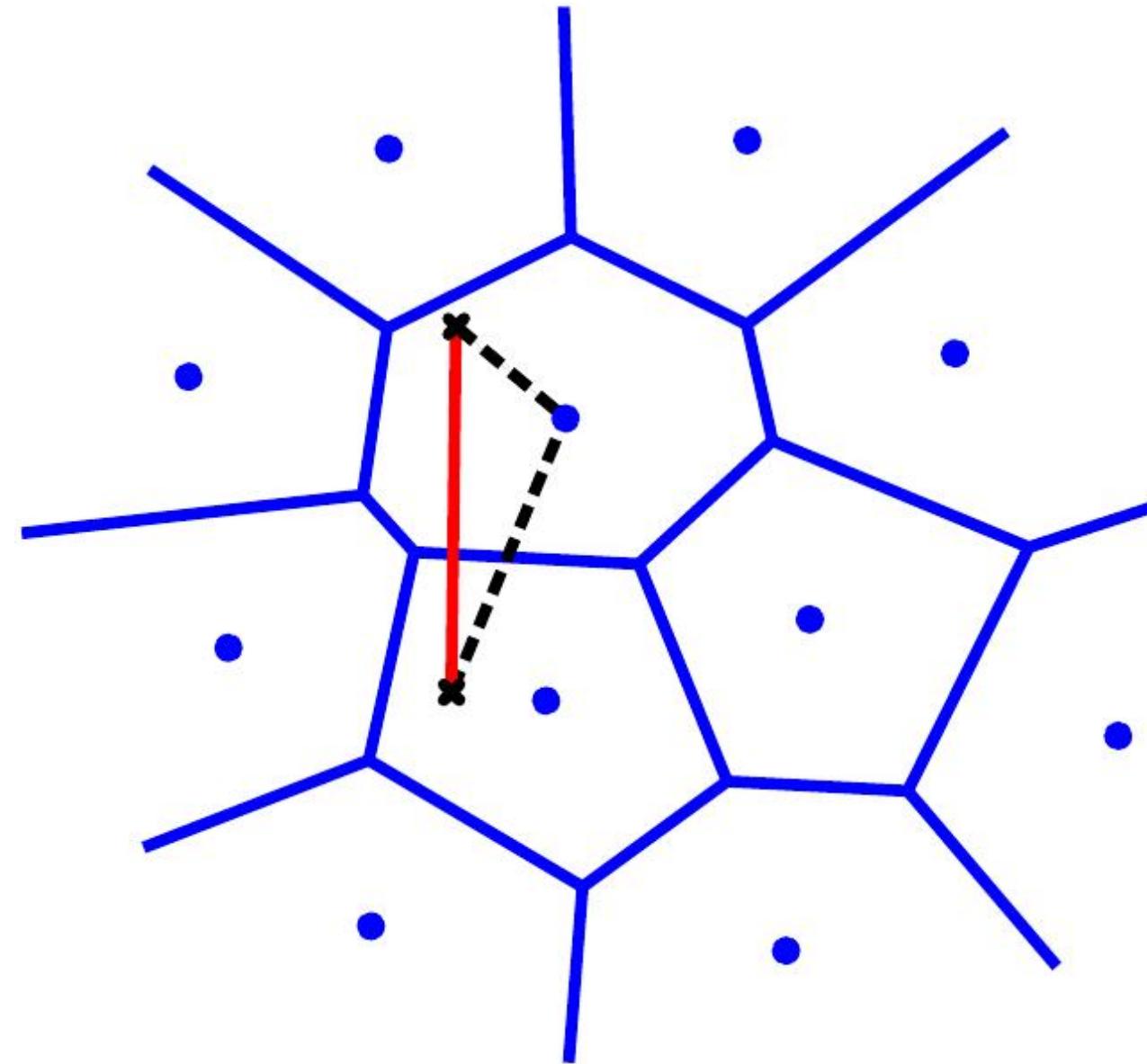


Quantization

Делим области на кластеры, и
каждую точку отождествляем с
центром ее кластера.

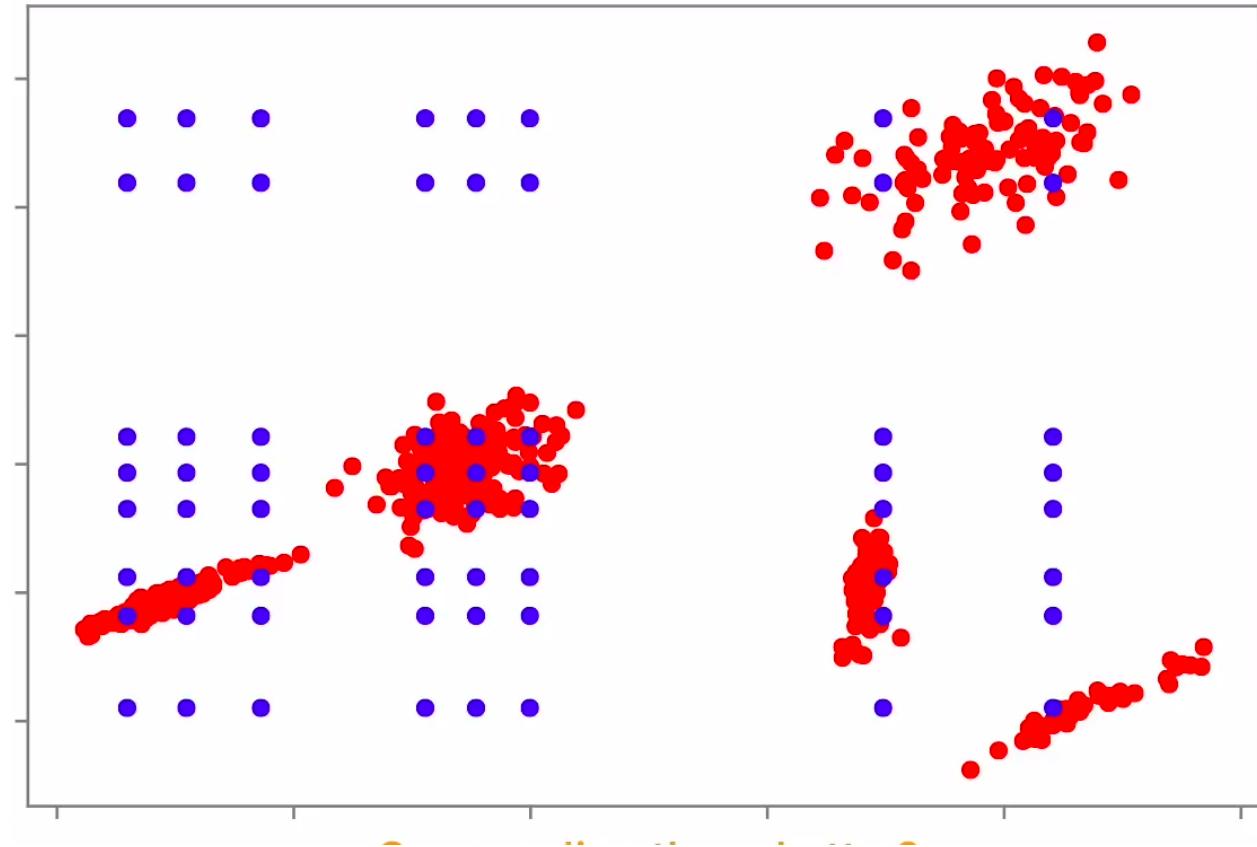


Потеря точности



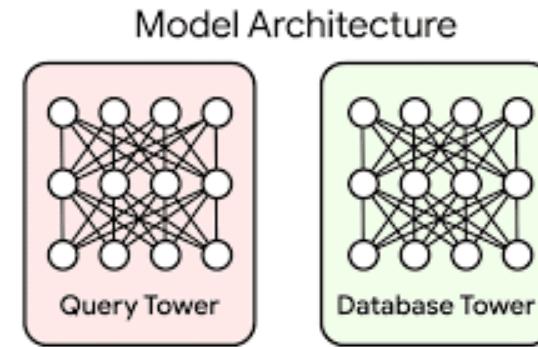
Увеличиваем количество кластеров

Кластеризуем по ординатно, за счет чего делаем большую сетку центров размера $O(N \times M)$, храня при этом $O(N + M)$ памяти

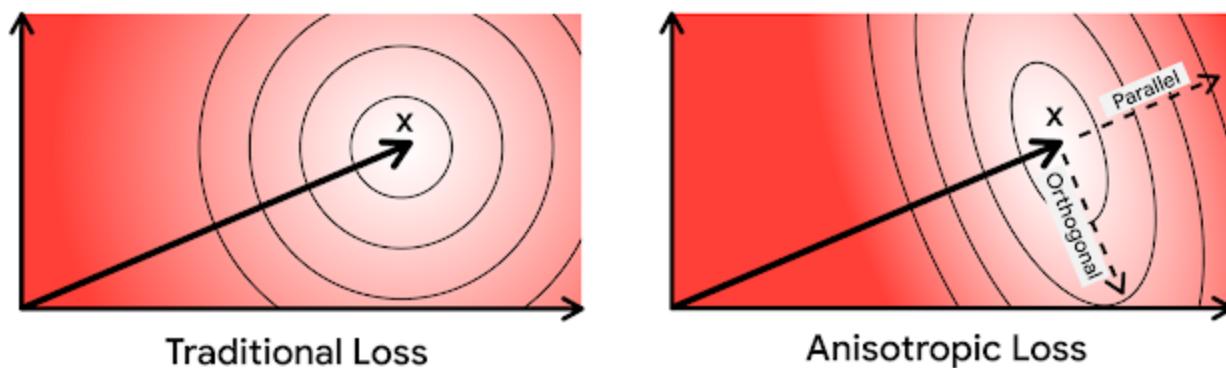
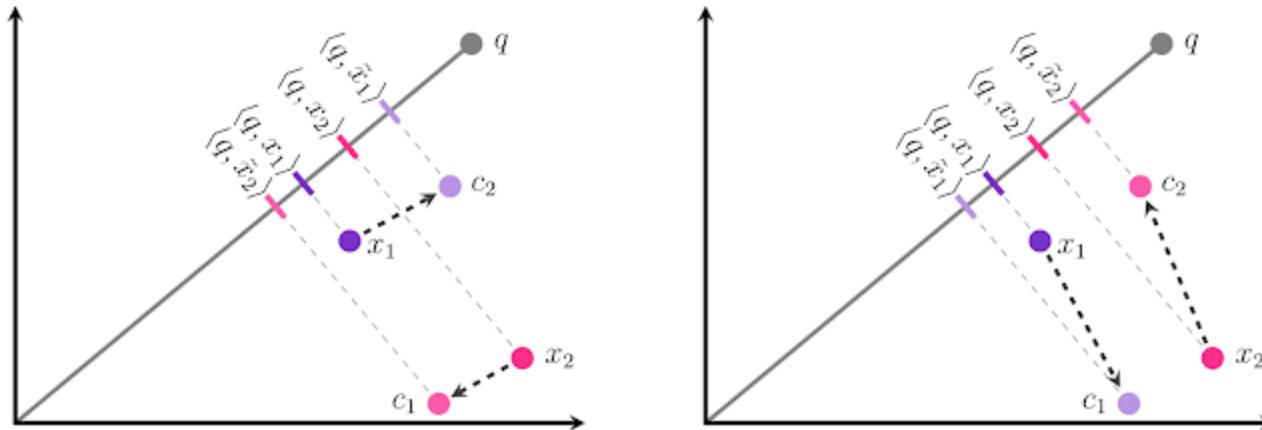


Scann

Цель такая же: найти эмбеддинг
и выделить центры кластеров.

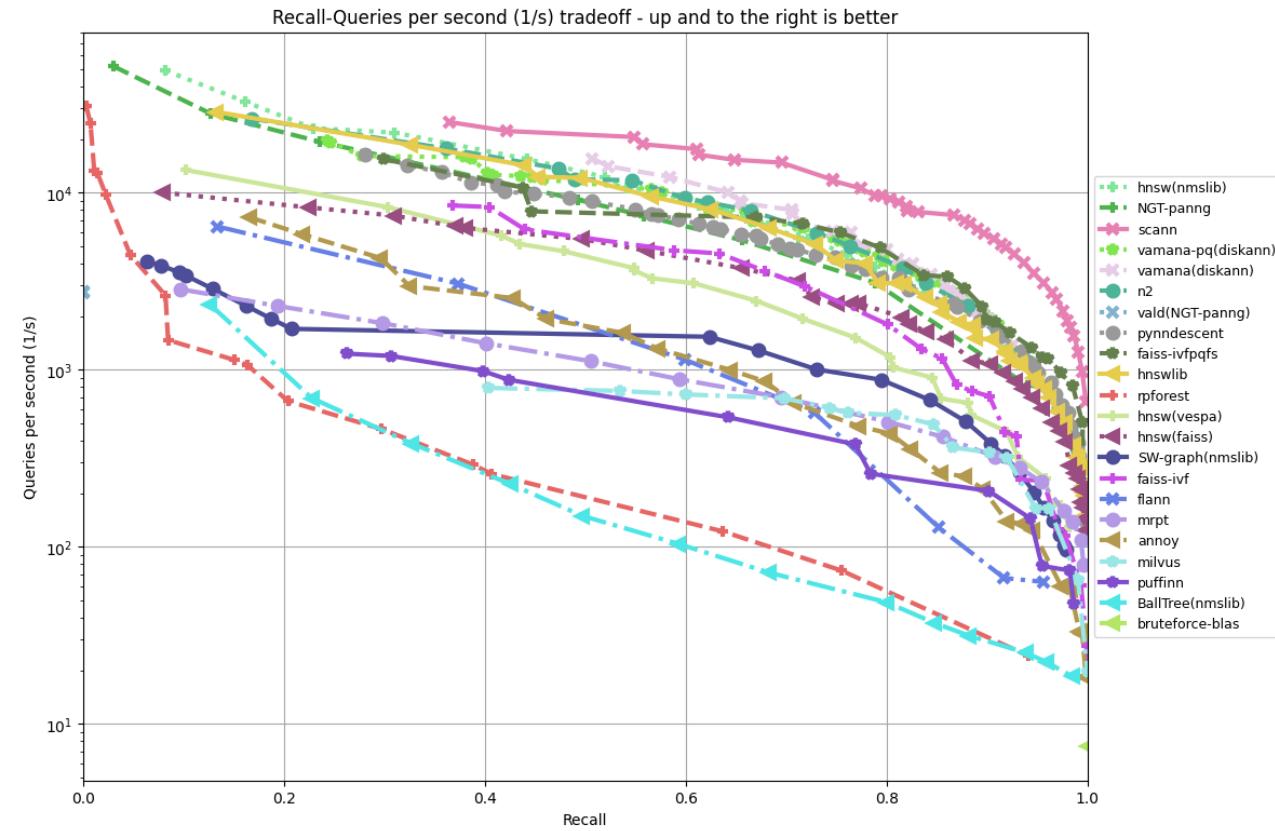


Оптимизация похожести скалярного произведения



Benchmark

Ссылка



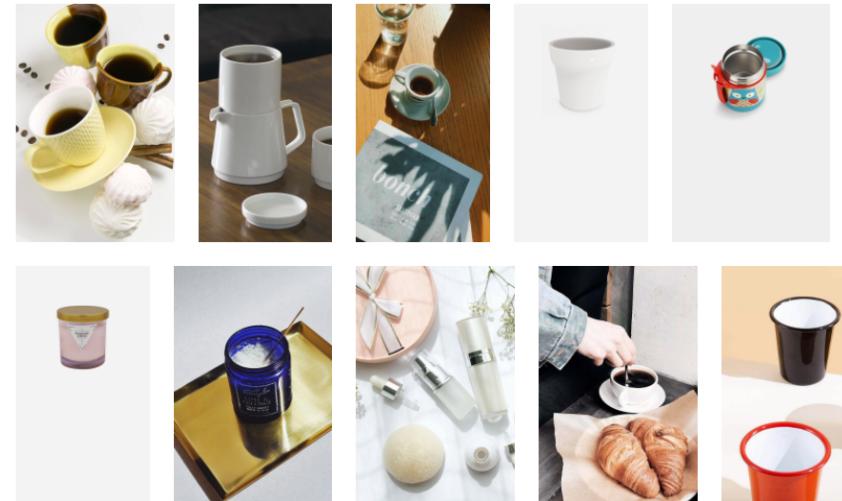
Пример приложения

SimSearch

Query:



Results:



Еще примеры

SimSearch

Query:



Results:



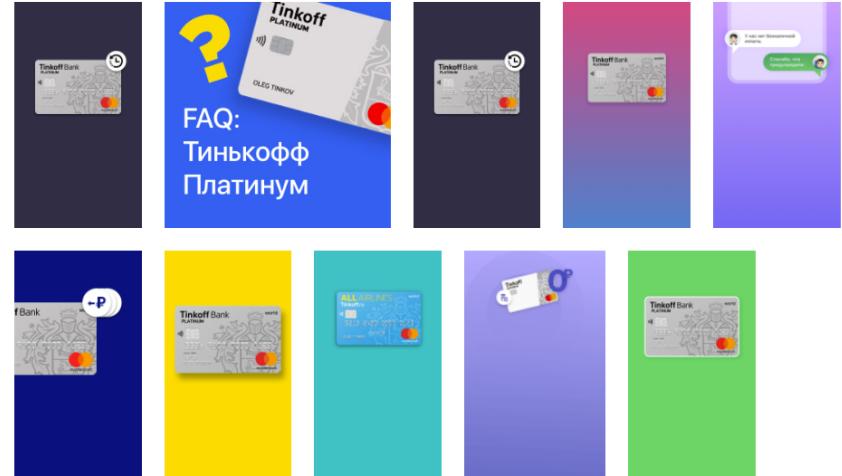
И еще

SimSearch

Query:



Results:



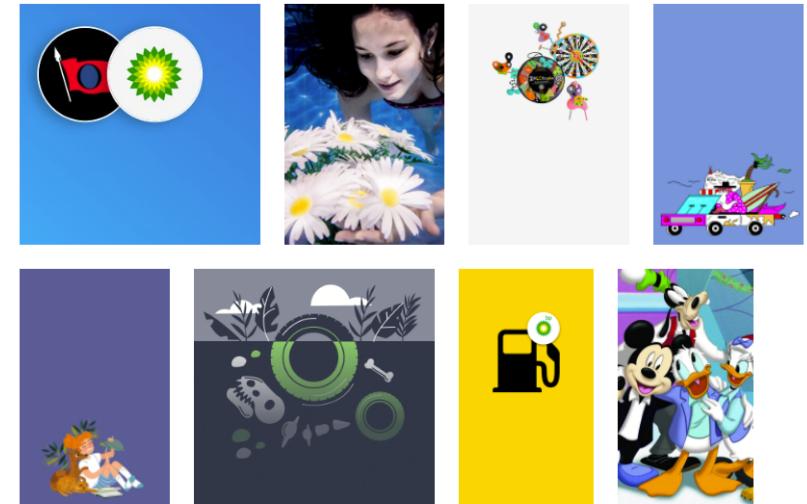
И напоследок

SimSearch

Query:



Results:



Если интересно

Больше про этот сервис можно почитать в [моих пдфках](#) на гитхабе

Дальнейшие планы

- Экзамен (до 9 мая включительно)
- Подводим итог по формам (в процессе)
- Открываем дорешку
- Стажировки, мерч, сертификаты и прочее - ждать информацию от Маши

**Всем спасибо за участие в
курсе!**

