Сравнительный анализ методов быстрого поиска ближайших соседей

Илья Федоров

Май 2020

Содержание

- Зачем это нужно?
- Структуры данных
- Почему деревья не работают
- Приближенные методы: LSH
- Приближенные методы: FAISS
- Приближенные методы: HNSW
- Сравнение
- Итоги

Зачем оптимизировать KNN?

- Очень большие датасеты
- Очень частые запросы
- Поиск дубликатов
- Неэффективность прямого перебора

Структуры данных

- KD-Tree
- Ball-tree
- Ball*-tree
- R-Tree
- etc...

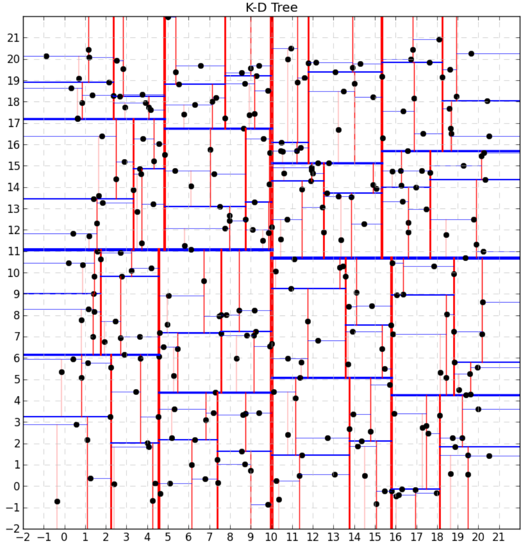
algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, optional*

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

Структуры данных



Сложности в пространствах высокой размерности

Почему это так? Проклятие размерности

Weber, Roger et al. "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces." VLDB (1998).

Сложности в пространствах высокой размерности

Observation 1 (Number of partitions)
The most simple partitioning scheme splits the data space in each dimension into two halves. With d dimensions, there are 2^d partitions. With $d \leq 10$ and N on the order of 10^6, such a partitioning makes sense. However, if d is larger, say $d = 100$, there are around 10^{30} partitions for only 10^6 points—the overwhelming majority of the partitions are empty.

Observation 2 (Data space is sparsely populated) Consider a hyper-cube range query with length l in all dimensions as depicted in Figure 1(a). The probability that a point lies within that range query is given by:

$$P^d[s] = s^d \quad (1)$$

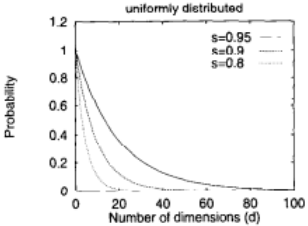


Figure 2: The probability function $P^d[s]$.

Сложности в пространствах высокой размерности

Observation 3 (Spherical range queries)
The largest spherical query that fits entirely within the data space is the query $sp^d(Q, 0.5)$, where Q is the centroid of the data space (see Figure 1(b)). The probability that an arbitrary point R lies within this sphere is given by the spheres volume:[1]

$$P[R \in sp^d(Q, \frac{1}{2})] = \frac{Vol(sp^d(Q, \frac{1}{2}))}{Vol(\Omega)} = \frac{\sqrt{\pi^d} \cdot (\frac{1}{2})^d}{\Gamma(\frac{d}{2}+1)} \quad (2)$$

If d is even, then this probability simplifies to

$$P[R \in sp^d(Q, \frac{1}{2})] = \frac{\sqrt{\pi^d} \cdot (\frac{1}{2})^d}{(\frac{d}{2})!} \quad (3)$$

Observation 4 (Exponentially growing DB size)
Given equation (2), we can determine the size a data set would have to have such that, on average, at least one point falls into the sphere $sp^d(Q, 0.5)$ (for even d):

$$N(d) = \frac{(\frac{d}{2})!}{\sqrt{\pi^d} \cdot (\frac{1}{2})^d} \quad (4)$$

d	$P[R \in sp^d(Q, 0.5)]$	$N(d)$
2	0.785	1.273
4	0.308	3.242
10	0.002	401.5
20	$2.461 \cdot 10^{-8}$	40'631'627
40	$3.278 \cdot 10^{-21}$	$3.050 \cdot 10^{20}$
100	$1.868 \cdot 10^{-70}$	$5.353 \cdot 10^{69}$

Table 2: Probability that a point lies within the largest range query inside Ω, and the expected database size.

Сложности в пространствах высокой размерности

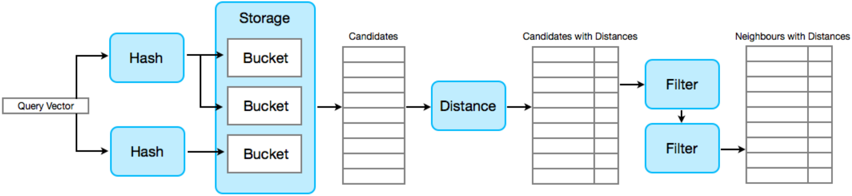
Conclusion 1 (Performance) *For any clustering and partitioning method there is a dimensionality $\hat{d}$ beyond which a simple sequential scan performs better. Because equation (17) establishes a crude estimation, in practice this threshold $\hat{d}$ will be well below 610.*

Conclusion 2 (Complexity) *The complexity of any clustering and partitioning methods tends towards $O(N)$ as dimensionality increases.*

Conclusion 3 (Degeneration) *For every partitioning and clustering method there is a dimensionality $\bar{d}$ such that, on average, all blocks are accessed if the number of dimensions exceeds $\bar{d}$.*

Приближенные методы: LSH

Идея: хешировать данные так, чтобы похожие элементы имели равные хэши, а сильно отличающиеся - различные

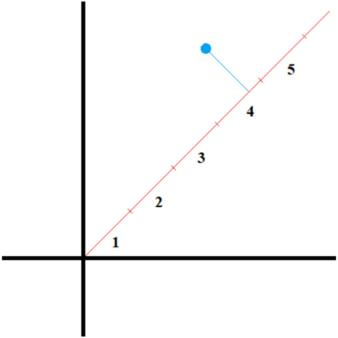


Приближенные методы: LSH

Для косинусного расстояния $h(x) = sign\langle w, x\rangle$

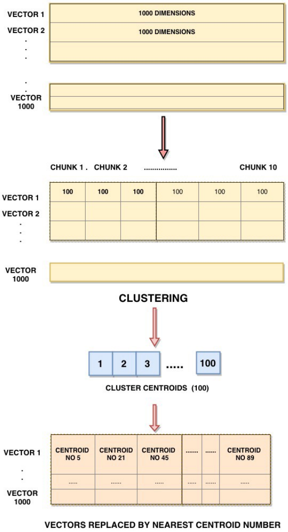
Для евклидового расстояния $h(x) = sign\langle w, x\rangle \ \ mod \ \ a$

$H(X) = (h_1(x), h_2(x), \ldots, h_k(x))$



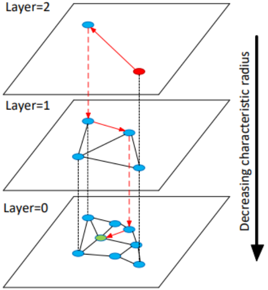
Приближенные методы: FAISS

Основная идея - product quantization

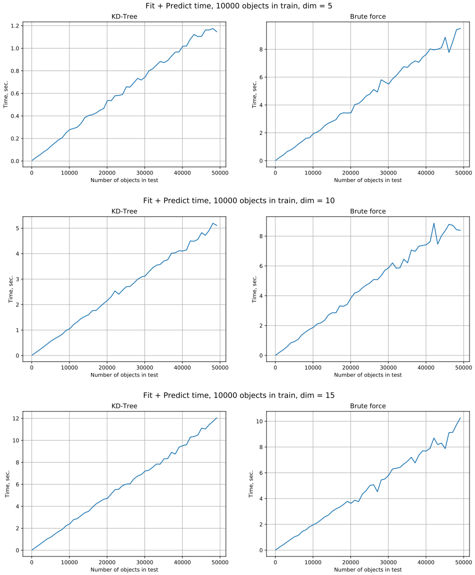


Приближенные методы: HNSW

Посмотрим систему графов и будем двигаться по ребрам, пока уменьшаем расстояние до запроса

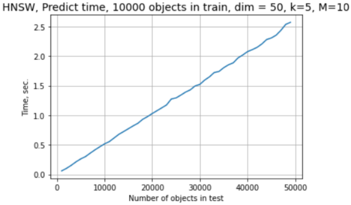


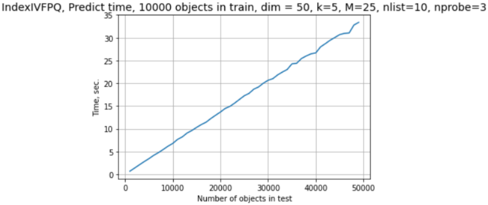
Сравнение

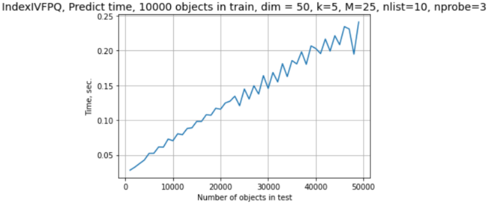


Fit + Predict time, 10000 objects in train, dim = 5
KD-Tree
Brute force
Time, sec.
Number of objects in test
Fit + Predict time, 10000 objects in train, dim = 10
KD-Tree
Brute force
Time, sec.
Number of objects in test
Fit + Predict time, 10000 objects in train, dim = 15
KD-Tree
Brute force
Time, sec.
Number of objects in test

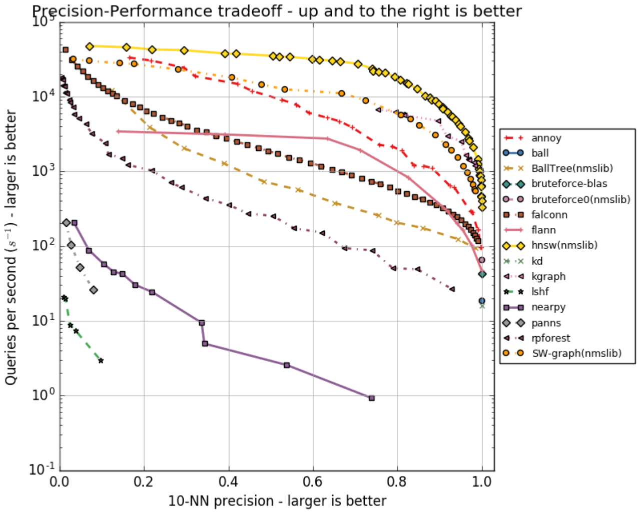
Сравнение


HNSW, Predict time, 10000 objects in train, dim = 50, k=5, M=10
Time, sec.
Number of objects in test


IndexIVFPQ, Predict time, 10000 objects in train, dim = 50, k=5, M=25, nlist=10, nprobe=3
Time, sec.
Number of objects in test


IndexIVFPQ, Predict time, 10000 objects in train, dim = 50, k=5, M=25, nlist=10, nprobe=3
Time, sec.
Number of objects in test

Сравнение



Precision-Performance tradeoff - up and to the right is better
Queries per second (s^{-1}) - larger is better
10-NN precision - larger is better
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
0.0
0.2
0.4
0.6
0.8
1.0
annoy
ball
BallTree(nmslib)
bruteforce-blas
bruteforce0(nmslib)
falconn
flann
hnsw(nmslib)
kd
kgraph
lshf
nearpy
panns
rpforest
SW-graph(nmslib)

Сравнение

Алгоритм	Преимущества	Недостатки
Прямой перебор	Простота	Неэффективность
kd-tree (и другие)	Есть в стандартных библиотеках	Неэффективен в пространствах высокой размерности
LSH	Гибкость	Проигрывает HNSW и FAISS в скорости и затратах на память
FAISS	Гибкость (множество параметров) Высокая эффективность на GPU Сжатые представления векторов	На CPU работает медленнее, чем HNSW
HNSW	Простота Высокая эффективность	Не поддерживает сжатие векторов

Итоги

Что было сделано:

- Написан подробный обзор классических и современных методов
- Установлен примерный порог размерности, при которой эффективны деревья
- Проведен ряд экспериментов для сравнения алгоритмов
- FAISS протестирован на GPU