Near-Neighbor Search at Scale

Casey Stella

March 4, 2012

Table of Contents

Introduction

- Hi, I'm Casey

Introduction

- Hi, I'm Casey
- I am a recovering Mathematician

Introduction

- Hi, I'm Casey
- I am a recovering Mathematician
- I work at Explorys

Introduction

- Hi, I'm Casey
- I am a recovering Mathematician
- I work at Explorys
- I am a senior engineer on the high speed indexes team

Introduction

- Hi, I'm Casey
- I am a recovering Mathematician
- I work at Explorys
- I am a senior engineer on the high speed indexes team
- I use Hadoop and HBase (a NoSQL datastore) every day

Introduction

- Hi, I'm Casey
- I am a recovering Mathematician
- I work at Explorys
- I am a senior engineer on the high speed indexes team
- I use Hadoop and HBase (a NoSQL datastore) every day
- I'm here to talk about Near-Neighbor Searches

Introduction

- Hi, I'm Casey
- I am a recovering Mathematician
- I work at Explorys
- I am a senior engineer on the high speed indexes team
- I use Hadoop and HBase (a NoSQL datastore) every day
- I'm here to talk about Near-Neighbor Searches
- In particular, about how they're hard to do efficiently at scale.

Near Neighbor Search

- Given a point in a vector space, find the nearest points according to some metric
  - You probably know a few, like $\mathbb{R}^2$ from Calculus
  - You probably know a metric like L_2, or the euclidean metric
  - Or L_1 a.k.a. Taxicab distance from A.I.
- Many problems can be rephrased as a near neighbor search (or use it as a primary component)
  - Recommendation Systems
  - Contextual Marketing (i.e. ads)
  - Clustering data
  - Lots more

Traditional Approach

- A naïve approach would be $O(n)$
- A less naïve approach typically involves *kd*-trees
- These tend to scale poorly in very high dimensions
  - The rule of thumb is for dimension k and number of points N, $N >> 2^k$[1]
  - Otherwise you end up doing a nearly exhaustive search most of the time
  - In these situations, approximation algorithms are typically used
- It's also not clear that they work for non-L_2 metrics

[1] Jacob E. Goodman, Joseph O'Rourke and Piotr Indyk (Ed.) (2004). "Chapter 39 : Nearest neighbours in high-dimensional spaces". Handbook of Discrete and Computational Geometry (2nd ed.). CRC Press.

"Big Data"

- Sometimes we have to deal with large amounts of data
- Traditionally we've put that data in SQL tables
- Scaling SQL databases is a pain in the ass
  - Explicit sharding breaks joins
  - Have to worry about node availability yourself
  - A lot of engineering work

Schema-less NoSQL data stores

- Recently there has been a movement to use distributed schema-less data stores instead
- These also happen to be a pain in the ass
- Conform to a map interface typically
  - put(Key k, Value v)
  - get(Key k)
  - delete(Key k)
- Examples of these are HBase, Cassandra, MongoDB, MemcacheDB
- It would be very nice to be able to use this to find nearest neighbors, but how?

Near Neighbor Searches

- Often we **need** high dimension (see previous talk)
- Often we have **many** points
- Often we'll accept an approximation
- Often we are looking for data in a fixed radius.

Locality Sensitive Hashing

- **Locality Sensitive Hashing** is a probabalistic technique to group "close" vectors according to a given metric.
- We can use this to group our "near" vectors into buckets
- You can construct multiple hash functions (i.e. families) and compose to increase the accuracy at the expense of runtime complexity
- You can use multiple LSH functions and put the same input data into each bucket, thereby increasing accuracy at the expense of space complexity

Locality Sensitive Hashing: The Cons

- Different hash function for different distance metrics (not all have them)
  - Exist for the biggest ones
  - L_k for $k > 0$, cosine-distance, min-hash, kernel-based metrics (i.e. machine learned distance metrics)
- Not all LSH functions have theoretical bounds about accuracy
  - Almost all research focuses on **nearest** neighbor searches
  - Practical alternative is to sample your data and measure

Stable Distributions and the L_k metric

- Based on the research of Piotr Indyk, et. al.[2]
- They found that 1-stable and 2-stable distributions could be used to construct families of locality sensitive hashes for L_1 and L_2 metrics
- What the hell are p-stable distributions?
  - If you draw a vector a from a p-stable distribution X, $a.(v_1 - v_2)$ is distributed exactly as $||v_1 - v_2||X$
  - Know that the Normal distribution is 2-stable and the Cauchy distribution is 1-stable
- Some intuition:

[2] Datar, M.; Immorlica, N., Indyk, P., Mirrokni, V.S. (2004). "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions". Proceedings of the Symposium on Computational Geometry.

Some Intuition

- Take the real number line and split it up into segments of length r, we can assign each segment an index and hash vectors into these segments.
- This should preserve locality because we're mapping $a.(v_1 - v_2)$ onto that segment
- Different choices of a make different functions with the same characteristics.
- If you don't understand, that's ok..it's not terribly obvious. You can treat this as a black box.

Spatial Search

- I've begun creating a simple library to assist in the use of these locality sensitive hashes
- It's Datastore agnostic
- L_1 and L_2 are implemented as well as a utility to assist in choosing parameters
- Next up is min-hash
- `https://github.com/cestella/SpatialSearch`

Conclusion

- Thanks for your attention
- Follow me on twitter @*casey_stella*
- Find me at
  - `http://caseystella.com`
  - `https://github.com/cestella`