SimSearch doc

1. Đặt vấn đề: Similarity Searching, statement, real-world problems
   1. Similarity Search

Similarity Search (SS) đề cập tới những kỹ thuật tìm kiếm ra một tập con các thực thể trong một tập thực thể ban đầu, với yêu cầu về sự tương tự với thực thể tìm kiếm (query entity). Ở những bài toán SS trong thực tế, gần như không thể so sánh thực thể tìm kiếm với toàn bộ tập thực thể và vét cạn những thực thể giống nó nhất mà phải dùng một số các phương pháp Machine Learning hoặc Deep Learning để rút ngắn thời gian tìm kiếm và đảm bảo tiêu chí khả mở của phương pháp tìm kiếm khi tập thực thể tăng lên hàng triệu thực thể. Trong các bài toán này, một trong những kiến thức thường thấy có sẵn là độ tương tự (similarity) giữa các thực thể với nhau, được tính toán bởi 1 độ đo tương đồng (similarity metrics) hoặc độ đo khoảng cách (distance metrics), tuy nhiên, kiến thức này thường có chi phí tính toán rất lớn.

* 1. Ứng dụng thực tế của SimSearch

Một trong những ứng dụng quen thuộc của Similarity Searching là pha đầu tiên trong các thuật toán Collaborative Filtering, khi một tập những người dùng (hoặc item) tương đồng với người dùng đang recommend, cần được trích xuất ra khỏi tập toàn bộ người dùng. Một trong những ứng dụng nữa của SS là tìm kiếm ảnh, tìm kiếm đoạn văn tương đồng trong các hệ thống tìm kiếm (search engines). Có rất nhiều ứng dụng của SS trong Sinh học với tìm kiếm đoạn gene hay tìm kiếm chuỗi protein,…

* 1. Đề cập tới bài toán SS, ta đặt các định nghĩa như sau

Cho 1 tập thực thể S, trong đó, có n các thực thể e, mỗi thực thể này được biểu diễn là 1 vector D chiều.

Cho 1 thực thể s\_q, có dạng biểu diễn giống với các thực thể e.

Cho 1 độ đo tương đồng (Similarity), hoặc một độ đo khoảng cách giữa các thực thể với nhau (Distance)

Yêu cầu tìm kiếm k thực thể có độ tương đồng cao nhất hoặc khoảng cách thấp nhất với e.

* 1. Các khó khăn của bài toán SS

Khi đề cập tới bài toán SS đã được phát biểu như trên, người đọc có thể thấy ngay hai trong những hướng mở rộng của bài toán: mở rộng về D và mở rộng về n. Nói cách khác là chi tiết hóa về biểu diễn thực thể và mở rộng độ lớn của không gian tìm kiếm. Ở trong thực tế thì điều thứ 2 luôn xảy ra. Một tập thực thể sẽ “nở” ra hàng trăm nghìn hoặc hàng triệu thực thể, đôi khi với độ biểu diễn chi tiết hơn, phát triển từ những vector 10 chiều tới những embeddings hàng ngàn chiều. Đây cũng là lý do mà tìm kiếm vét cạn sẽ được loại bỏ khi nghiên cứu về những phương pháp SS cho ứng dụng thực tế.

Trong nghiên cứu lý thuyết về SS, tồn tại một ý tưởng là thay vì tìm kiếm chính xác tất cả các thực thể có độ tương đồng cao nhất, ta tìm kiếm xấp xỉ tập con này (ASS-Approximate Similarity Searching), hy sinh độ chính xác tối đa để gia tăng kết quả tìm kiếm. Cách làm này không đảm bảo sẽ tìm thấy k thực thể giống thực thể tìm kiếm nhất, nhưng sẽ tìm thấy phần lớn trong số chúng, với 1 vài thực thể sai (false positives).

Hai trong những hướng giải quyết bài toán SS đi theo ý tưởng này được đề cập ở đây là Product Quantization và Locality Sensitive Hashing. Cả 2 phương pháp này đều có một điểm chung là mong muốn rút ngắn thời gian tìm kiếm bằng cách sử dụng các kỹ thuật giảm số chiều của vector biểu diễn thực thể và thu hẹp không gian tìm kiếm, một cách đồng thời.

1. FAISS

Ra đời vào năm 2017, thư viện C++ (đi kèm Python Bindings) của Facebook là một trong những công cụ hữu hiệu để xử lý bài toán SS. Nền tảng của thư viện này là thuật toán Product Quantization, cùng với cài đặt tối ưu của nó trên các GPU. Với nền tảng lý thuyết và cài đặt tối ưu này, FAISS tự hào là một trong những công cụ vừa nhanh nhất để SS, vừa không hy sinh độ chính xác quá nhiều trong quá trình tìm kiếm.

* 1. Helloworld cùng FAISS
  2. Nền tảng thuật toán: Product Quantization

Do độ tương đồng của phương pháp tìm kiếm này với tìm kiếm k lân cận gần nhất (Appx. Nearest Neighbor) nên nó sẽ được đề cập như một trong những cách giải bài toán Nearest Neighbor Search.

* + Nén dữ liệu với vector quantization (VQ)

Xét một bộ dữ liệu S chứa n vector e, mỗi vector có thể được biểu diễn theo D chiều, ta có thể “nén” các biểu diễn dữ liệu này bằng cách sử dụng K-Means và tạo các Vonoroi cells và thay vì sử dụng trực tiếp biểu diễn D chiều, ta sử dụng tâm của một centroid gần nhất với mỗi e trong S:

q\_e = q(e) = C\_argmin(x, KMeans(y))

Giờ đây số lượng biểu diễn của n vector đã giảm xuống còn c vector - số lượng centroid có thể khi đã sử dụng KMeans cho bộ dữ liệu.

* + Short-code

Trong thực tế, chỉ 1 bộ centroid KMeans n chiều vẫn gây ra sự tốn kém trong các phương trình tính toán khoảng cách (sau này) của chúng ta. Vì vậy, thay vào đó, chúng ta sẽ chia nhỏ vector ban đầu ra thành m bộ vector con với độ dài mỗi vector là D/m.

e = [e\_1, e\_2, e\_3,…, e\_m]

Sau đó áp dụng K-Means clustering cho từng “bộ dữ liệu” các vector con, và lưu lại index (số chỉ) của centroid vào mỗi vị trí các vector con để thay thế cho vector con đó, điều này tạo thành một bộ dữ liệu các vector gọi là short-code, mỗi short-code có số chiều m. Bên cạnh đó, ta cũng lưu thông tin các centroid của mỗi thuật toán KMeans đã sử dụng về một bảng look-up để thực hiện tìm kiếm sau này. Đây là phương pháp Product Quantization.

* Khoảng cách đối xứng (SDC) và khoảng cách bất đối xứng (ADC)

Sau khi đã “giảm” độ phức tạp của bộ vector ban đầu qua phương pháp PQ, ta sẽ thu được một bộ vector đã giảm chiều đáng kể so với bộ vector ban đầu. Câu hỏi đặt ra ở đây là, chuyện gì sẽ xảy ra khi ta có nhu cầu tìm kiếm (search) k vector có độ tương quan cao nhất, trong bộ dữ liệu gốc, so với một vector query bất kỳ x, có số chiều bằng với vector gốc (D-chiều).

Một khoảng cách của vector x có D chiều bất kỳ, so với 1 vector y bất kỳ trong S sẽ có thể được tính toán theo 2 cách: sử dụng PQ lên x rồi so sánh với q(y) (đối xứng) hoặc so sánh trực tiếp x với q(y) (bất đối xứng). Hai cách này có chi phí tính toán tương tự nhau.

+ Khoảng cách đối xứng

* 1. Nền tảng thuật toán: IVF ADC
     1. ADC
     2. IVF
  2. Làm sao để ứng dụng FAISS trong hệ thống thực tế: Cảm quan và đánh giá của tôi.

1. LSH
   1. Helloworld cùng LSH trên PySpark
   2. Nền tảng thuật toán: LSH cho câu văn cùng Shingling và MinHash
   3. Nền tảng thuật toán: LSH cho các distance metrics khác(và các kiểu dữ liệu tương ứng)
   4. Làm sao để ứng dụng LSH cho các kiểu dữ liệu khác với PySpark.
2. Kết luận & Hướng phát triển kế tiếp
3. References.