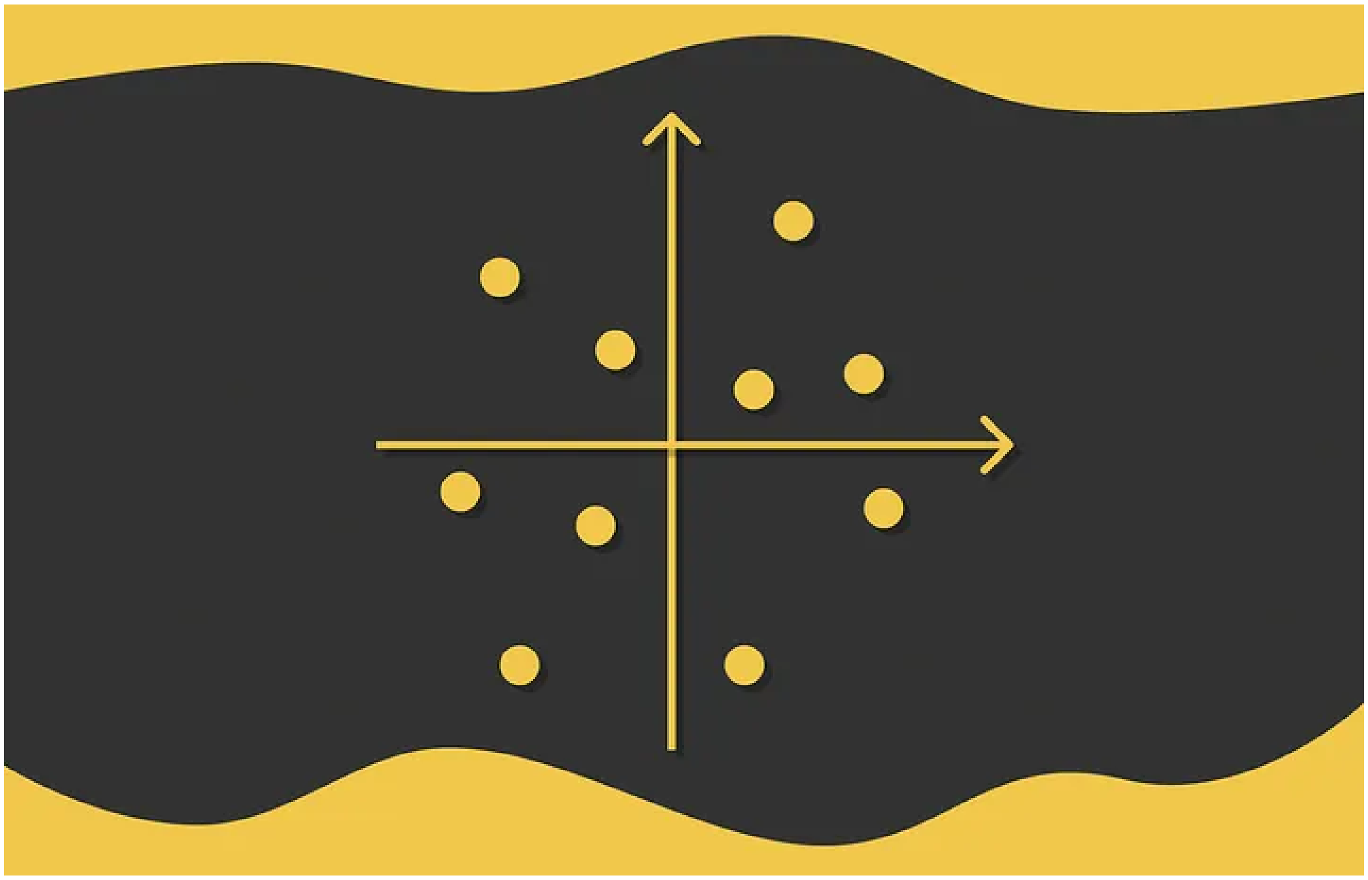
Search Medium Write

Similarity Search, Part 3: Blending Inverted File Index and Product Quantization

Discover how to combine two basic similarity search indexes to get the advantages of both

Vyacheslav Efimov · Following
Published in Towards Data Science · 8 min read · May 19

128



Similarity search is a problem where given a query the goal is to find the most similar documents to it among all the database documents.

Introduction

In data science, similarity search often appears in the NLP domain, search engines or recommender systems where the most relevant documents or items need to be retrieved for a query. There exists a large variety of different ways to improve search performance in massive volumes of data.

In the first two parts of this series we have discussed two fundamental algorithms in information retrieval: **inverted file index** and **product quantization**. Both of them optimize search performance but focus on different aspects: the first one accelerates the search speed while the latter compresses vectors to a smaller, memory-efficient representation.

Similarity Search, Part 1: kNN & Inverted File Index
Similarity search is a popular problem where given a query Q we need to find the most similar documents to it among all…
towardsdatascience.com

Similarity Search, Part 2: Product Quantization
In the first part of this article series, we looked at kNN and inverted file index structure for performing similarity…
medium.com

Since both algorithm focus on different aspects, the question that naturally arises is whether it is possible to merge these two algorithms into a new one

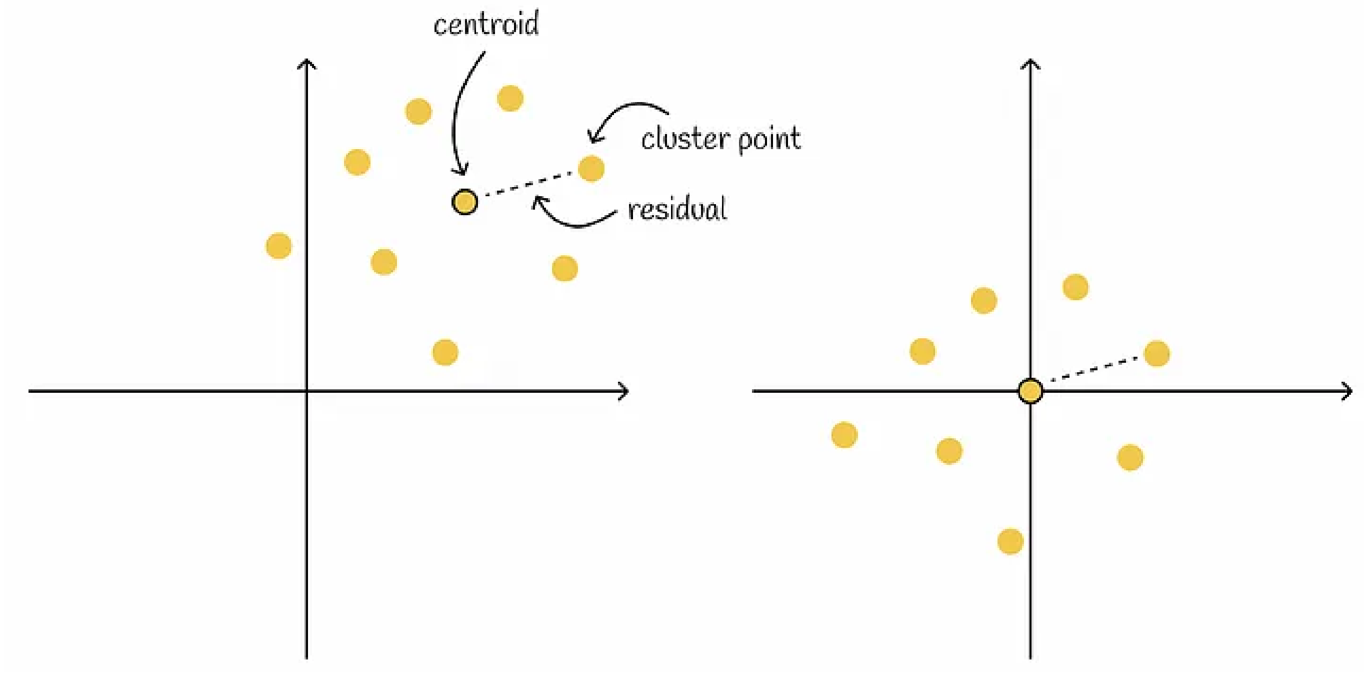
In this article, we will combine the advantages of both approaches to produce a fast and memory-efficient algorithm. For information, most of the discussed ideas will be based on this paper.

Before diving into details, it is necessary to understand what residual vectors are and get a simple intuition on their useful properties. We will use them later while designing an algorithm.

Residual vectors

Imagine a clustering algorithm was executed and it produced several clusters. Each cluster has a centroid and points associated with it. **Residual** is an offset of a point (vector) from its centroid. Basically, to find a residual for a particular vector, the vector has to be subtracted from its centroid.

If the cluster was built by the k-means algorithm, then the cluster centroid is the mean of all points belonging to that cluster. Thus, finding a residual from any point would be equivalent to subtracting the mean of a cluster from it. By subtracting the mean value from all points belonging to a particular cluster, the points would be centered around 0.



An original cluster of points is shown on the left. Then the cluster centroid is subtracted from all cluster points. The resulting residual vectors are shown on the right.

We can observe a useful fact:

> Replacing original vectors with their residuals does not change their relative position to each other.

That is, the distance between vectors stays always the same. Let us simply look at two equations below.

$$\text{distance}(x, y) = \sqrt{\sum_{i}^{n} (x_i - y_i)^2}$$

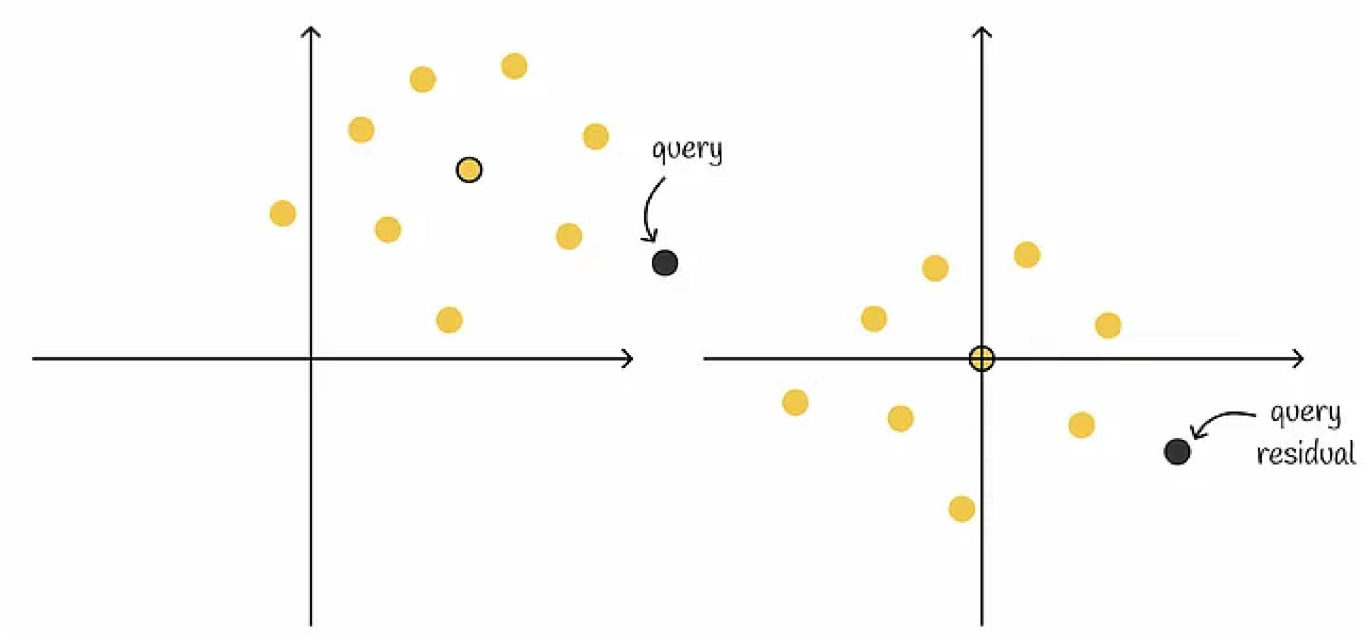
$$\text{distance}(x - c, y - c) = \sqrt{\sum_{i}^{n} ((x_i - c) - (y_i - c))^2} = \sqrt{\sum_{i}^{n} (x_i - y_i)^2}$$

Subtracting the mean does not change the relative distance

The first equation is the formula of euclidean distance between a pair of vectors. In the second equation, the cluster mean value is subtracted from both vectors. We can see that the mean term simply cancels out — the whole expression becomes identical to euclidean distance in the first equation!

> *We proved this statement by using the formula for the L2 metric (euclidean distance). It is important to remember that this rule may not work for other metrics.*

So, if for a given query the goal is to find the nearest neighbour, it is possible to just subtract the cluster mean from the query and proceed to the normal search procedure among residuals.

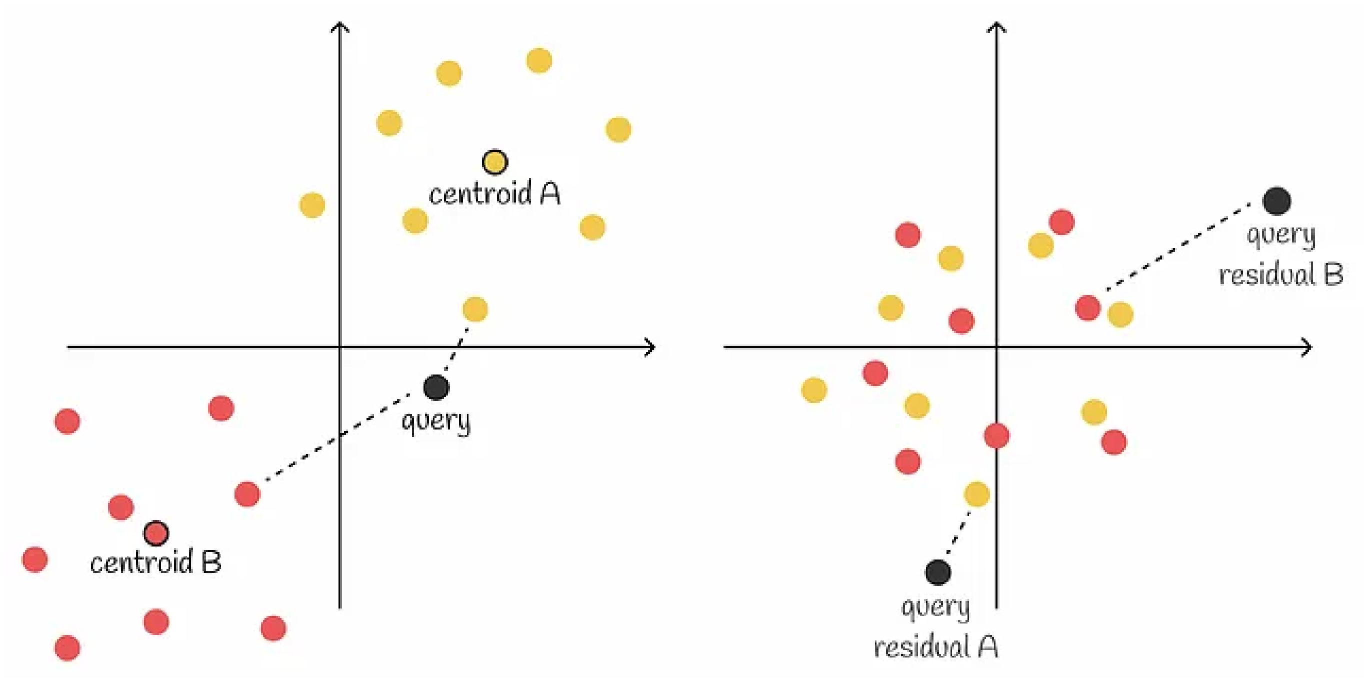


Subtracting a mean from a query does not change its relative position to other vectors

Right now let us look at another example in the figure below with two clusters where residuals for vectors of each cluster are calculated separately.

> Subtracting the mean of the corresponding centroid from each vector of a cluster will center all the dataset vectors around 0.

That is a useful observation that will be used in the future. Additionally, for a given query, we can calculate query residuals to all the clusters. The query residuals allow us to calculate distances to the original residuals of the cluster.



After subtracting mean values from each cluster, all the points become centered around 0. Relative position from query and query residuals to other points of corresponding clusters does not change.

Training

Having taken into considerations useful observations in the previous section, we can start designing the algorithm.

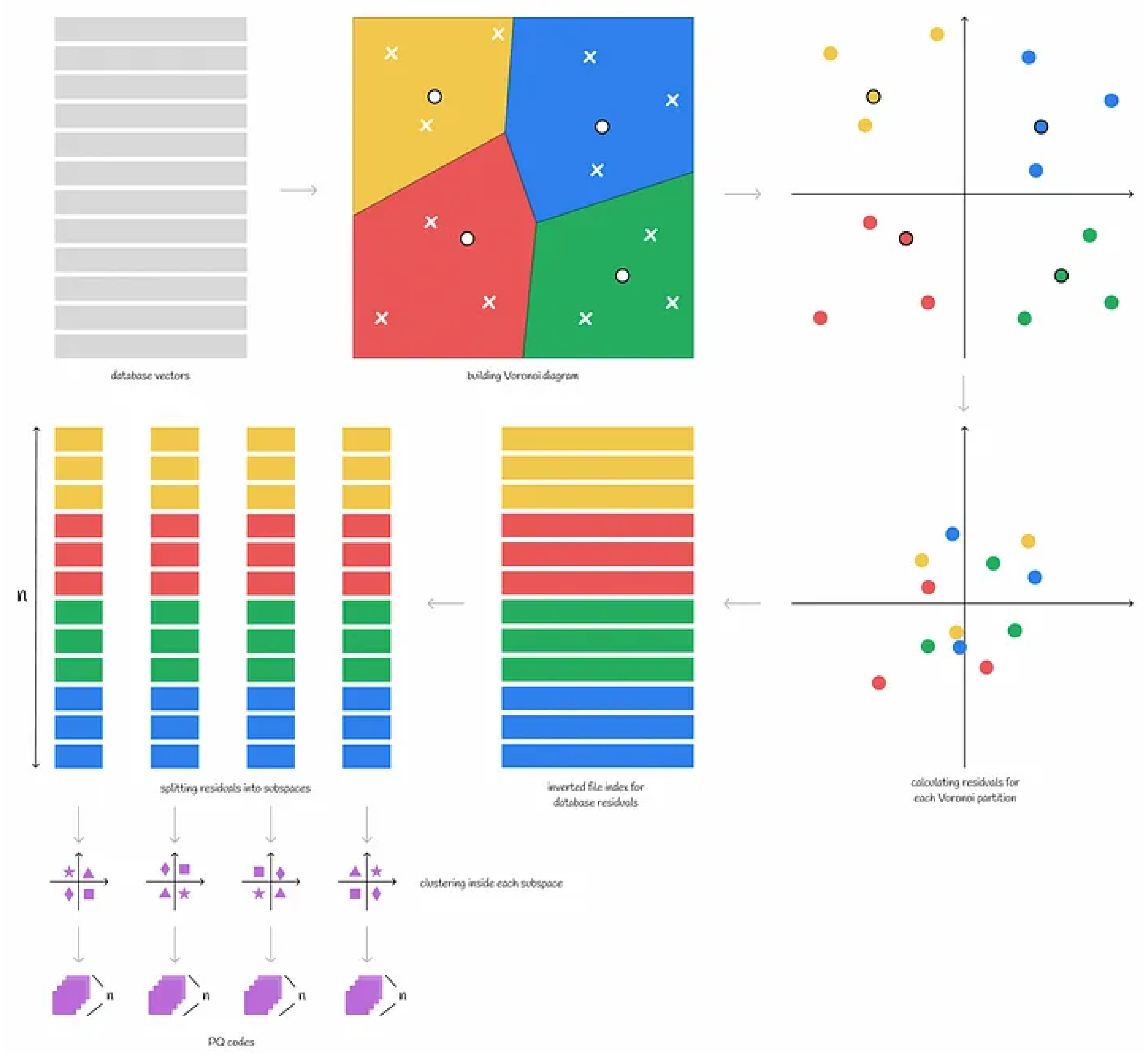
Given a vector database, an inverted file index is constructed that divides the set of vectors into *n* Voronoi partitions and thus reduces the search scope during the inference.

Inside each Voronoi partition, the coordinates of the centroid are subtracted from each vector. As a result, vectors from all the partitions become closer to each other and centered around 0. At this moment, there is no need the original vectors as we store their residuals instead.

After that, the product quantization algorithm is run on vectors from all the partitions.

Important aspect: the product quantization is **not** executed for each partition separately — that would be inefficient because the number of partitions usually tends to be high which will could result in a lot of memory needed to store all the codebooks. Instead, **the algorithm is executed for all the residuals from all partitions simultaneously**.

Effectively, now each subspace contains subvectors from different Voronoi partitions. Then, for each subspace, a clustering algorithm is performed and *k* clusters with their centroids are built, as usual.


database vectors
building Voronoi diagram
calculating residuals for each Voronoi partition
inverted file index for database residuals
n
splitting residuals into subspaces
clustering inside each subspace
n
PQ codes

Training process

What was the importance of replacing vectors with their residuals? If vectors were not replaced with their residuals, then each subspace would contain more various subvectors (because subspaces would store subvectors from different non-intersecting Voronoi partitions which could have been very far from each other in space). Now vectors (residuals) from different partitions overlap with each other. Since now there is less variety in each subspace, it takes fewer reproduction values to effectively represent vectors. In other words:

> With the same length of PQ codes used before, vectors can be represented more accurately because they have less variance.

Inference

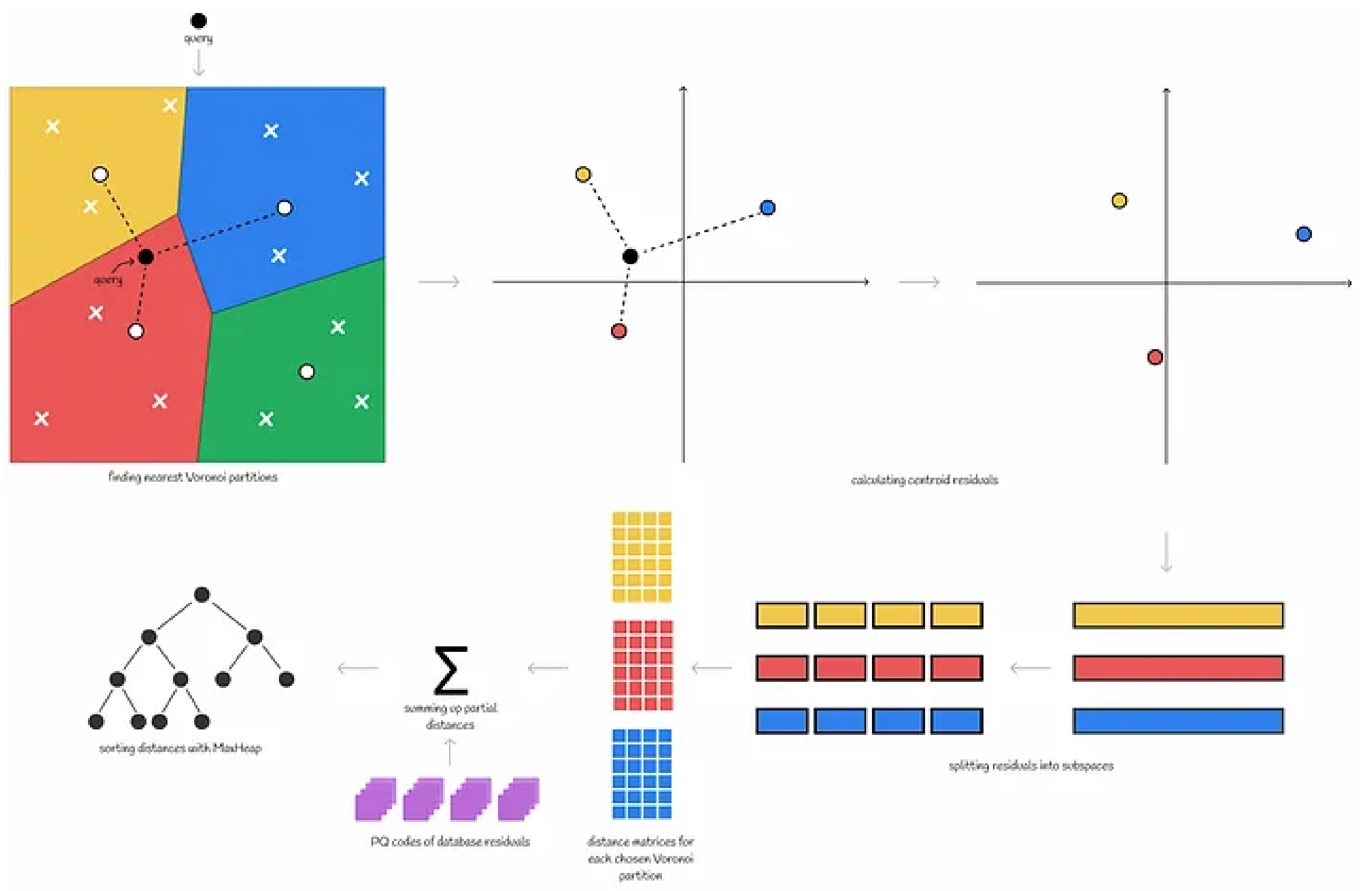
For a given query, the *k* nearest centroids of Voronoi partitions are found. All the points inside these regions are considered as candidates. Since the original vectors were initially replaced by their residuals in each Voronoi region, the residual of the query vector also needs to be calculated. In this case, the query residual needs to be calculated separately for every Voronoi partition (because every region has different centroids). Only residuals from chosen Voronoi partitions are going to the candidates.

The query residual is then split into subvectors. As in the original product quantization algorithm, for each subspace, the distance matrix *d* containing distances from subspace centroids to the query subvector is computed. It is essential to keep in mind that the query residual is different for each Voronoi partition. Basically it means that the distance matrix *d* needs to be computed separately for each of the query residuals. That is the computation price for the desired optimization.

Finally, partial distances are summed up as it was previously done in the product quantization algorithm.

Sorting results

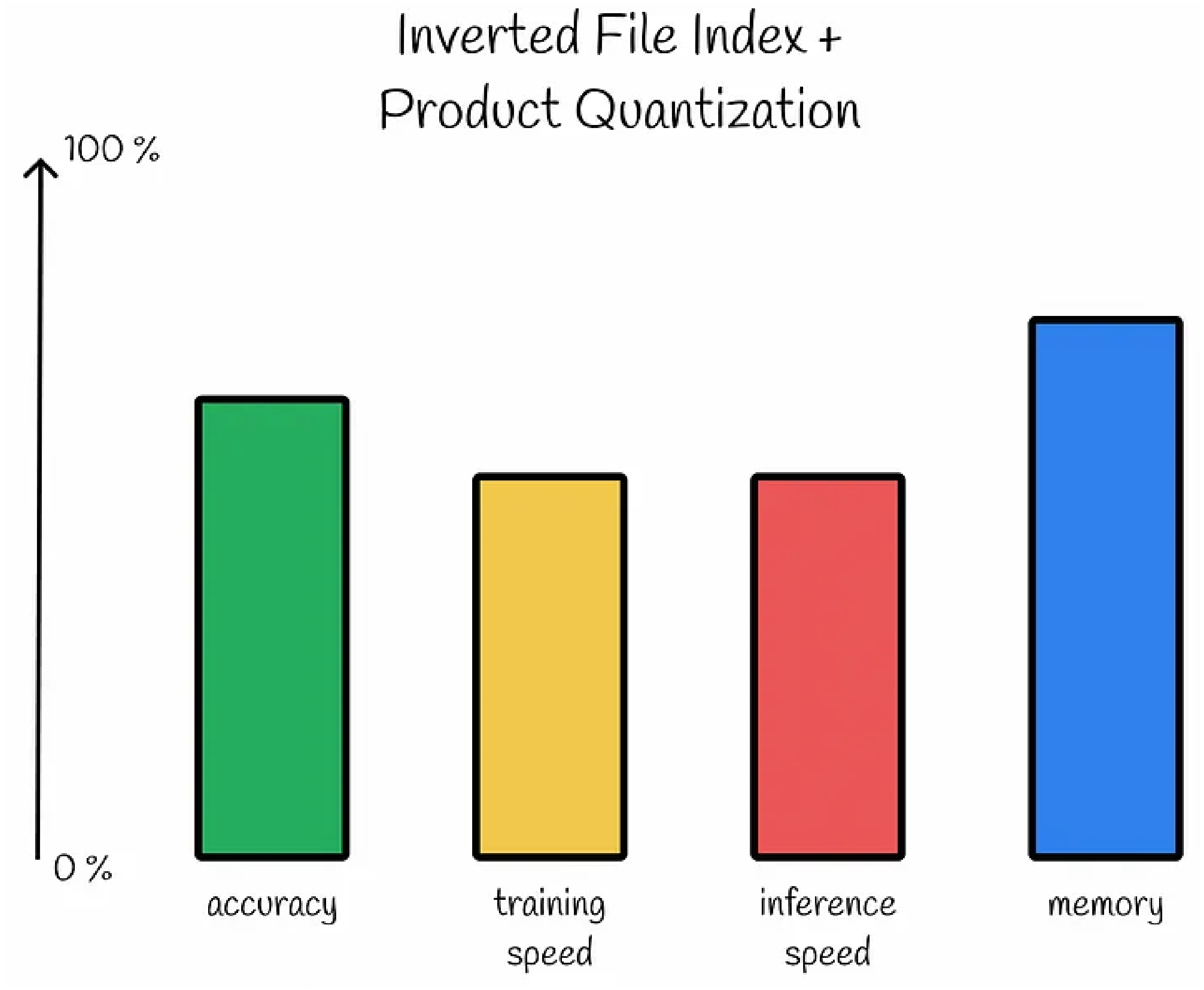
After all the distances are calculated, the *k* nearest neighbors need to be selected. To do it efficiently, authors propose maintaining a MaxHeap data structure. It has a limited capacity of *k* and at each step, it stores *k* current smallest distances. Whenever a new distance is calculated, its value is added to the MaxHeap only if the computed value is less than the largest value in the MaxHeap. After calculating all the distances, the answer to the query is already stored in the MaxHeap. The advantage of using the MaxHeap is its fast building time which is *O(n)*.


query
query
finding nearest Voronoi partitions
calculating centroid residuals
splitting residuals into subspaces
distance matrices for each chosen Voronoi partition
summing up partial distances
PQ codes of database residuals
sorting distances with MaxHeap

Inference process

Performance

The algorithm takes advantage of both inverted file index and product quantization. Depending on the number of Voronoi partitions probe during the inference, the same number of subvector-to-centroid matrices *d* needs to be computed and stored in the memory. This might look like a downside but comparing it to overall advantages, it is a pretty good trade-off.



The algorithm inherits a good search speed from inverted file index and compression efficiency from product quantization

Faiss implementation

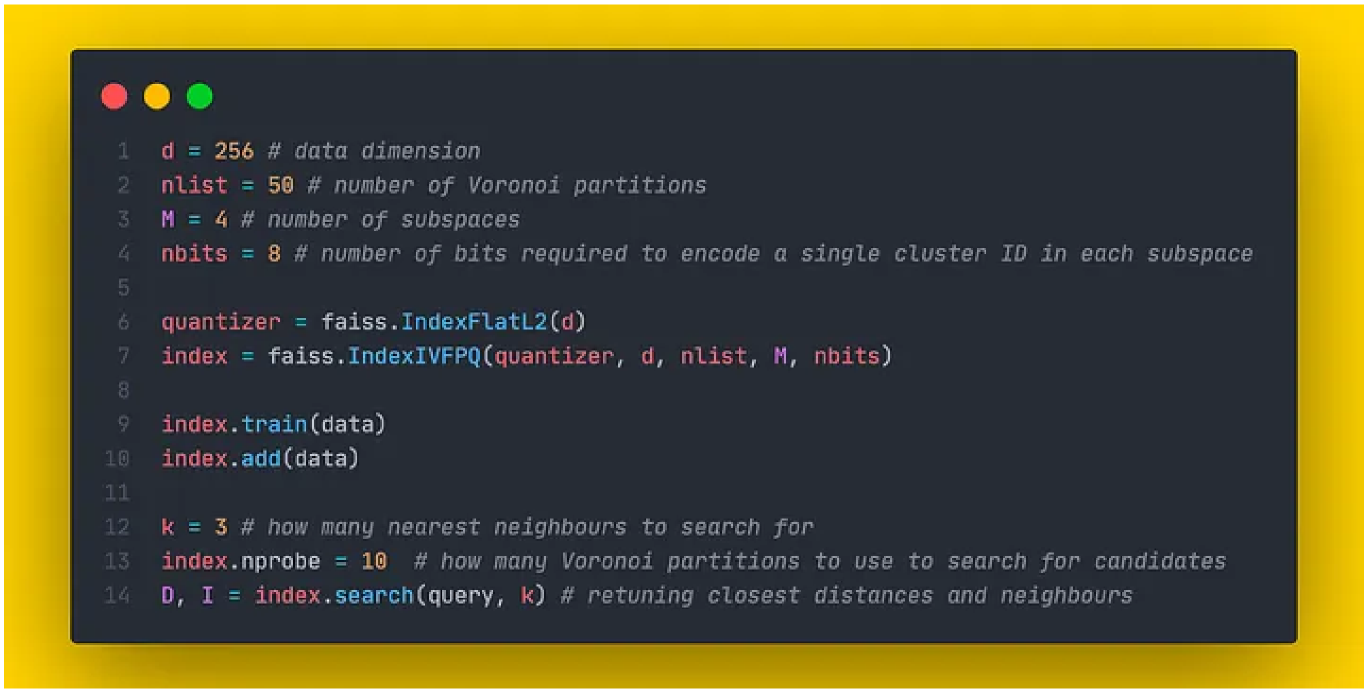
> *Faiss (Facebook AI Search Similarity) is a Python library written in C++ used for optimised similarity search. This library presents different types of indexes which are data structures used to efficiently store the data and perform queries.*

Based on the information from the Faiss documentation, we will see how inverted file and product quantization indexes can be combined together to form a new index.

Faiss implements the described algorithm in the *IndexIVFPQ* class which accepts the following arguments:

- **quantizer:** specifies how distance between vectors is computed.
- **d:** data dimensionality.
- **nlist:** number of Voronoi partitions.
- **M:** number of subspaces.
- **nbits:** number of bits it takes to encode a single cluster ID. This means that the number of total clusters in a single subspace will be equal to *k = 2^nbits*.

Additionally, it is possible to adjust the **nprobe** attribute which specifies how many Voronoi partitions must be used for the search of candidates during inference. Changing this parameter does not require retraining.



```python
d = 256 # data dimension
nlist = 50 # number of Voronoi partitions
M = 4 # number of subspaces
nbits = 8 # number of bits required to encode a single cluster ID in each subspace

quantizer = faiss.IndexFlatL2(d)
index = faiss.IndexIVFPQ(quantizer, d, nlist, M, nbits)

index.train(data)
index.add(data)

k = 3 # how many nearest neighbours to search for
index.nprobe = 10  # how many Voronoi partitions to use to search for candidates
D, I = index.search(query, k) # retuning closest distances and neighbours
```

Faiss implementation of IndexIVFPQ

Memory required to store a single vector is the same as in the original product quantization method except now we add 8 more bytes to store information about the vector in the inverted file index.

$$bytes = \left\lceil \frac{M * nbits}{8} \right\rceil + 8$$

Conclusion

Using the knowledge from the previous article parts, we have walked through the implementation of a state-of-the-art algorithm that achieves high memory compression and accelerated search speed. This algorithm is widely used in information retrieval systems when dealing with massive volumes of data.

Resources

- Product quantization for the nearest search
- Faiss documentation
- Faiss repository
- Summary of Faiss indexes

All images unless otherwise noted are by the author.

Similarity Search Inverted Index Quantization Index Machine Learning

More from the list: "NLP"

Curated by Himanshu Birla

Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

✦ · 11 min read · Sep 4, 2021

Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

✦ · 6 min read · Sep 3, 2021

Jon Gi... in

The Word2ve

✦ · 15 min rea

View list

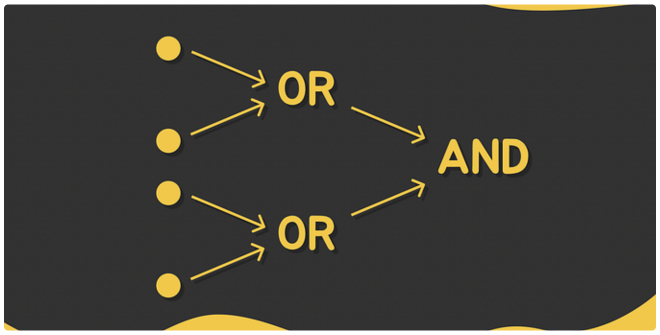
Written by Vyacheslav Efimov

Following

470 Followers · Writer for Towards Data Science

BSc in Software Engineering. Passionate machine learning engineer. Writer at Towards Data Science.

More from Vyacheslav Efimov and Towards Data Science



Vyacheslav Efimov in Towards Data Science

Similarity Search, Part 7: LSH Compositions

Dive into combinations of LSH functions to guarantee a more reliable search

11 min read · Jul 24

43

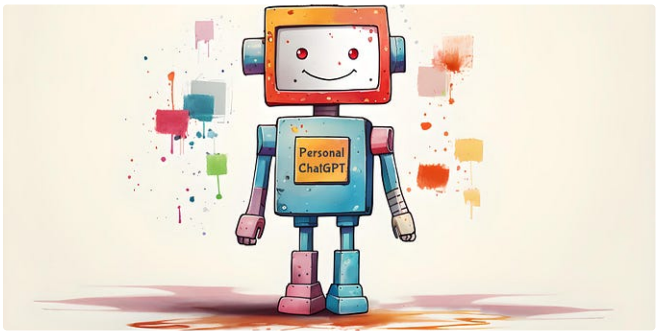
Antonis Makropoulos in Towards Data Science

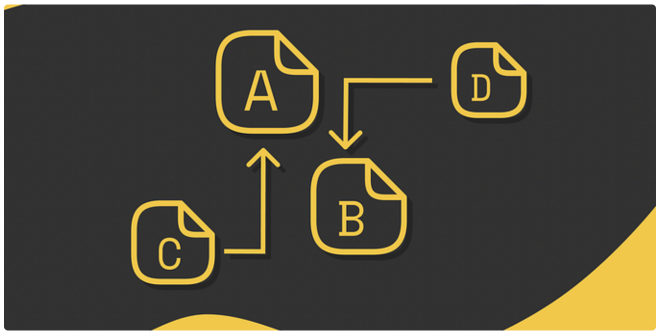
How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and...

10 min read · Sep 17

549 11





Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

· 15 min read · Sep 8

595 7

Vyacheslav Efimov in Towards Data Science

Visualised Explanation of PageRank

Discover how Google search engine ranks documents based on their link structure

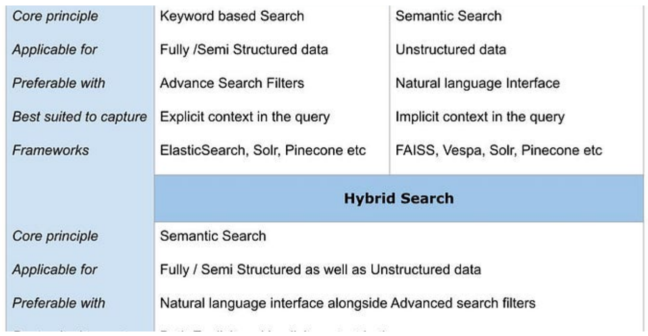
14 min read · Aug 10

40 1

See all from Vyacheslav Efimov

See all from Towards Data Science

Recommended from Medium



Core principle	Keyword based Search	Semantic Search
Applicable for	Fully /Semi Structured data	Unstructured data
Preferable with	Advance Search Filters	Natural language Interface
Best suited to capture	Explicit context in the query	Implicit context in the query
Frameworks	ElasticSearch, Solr, Pinecone etc	FAISS, Vespa, Solr, Pinecone etc
	Hybrid Search	
Core principle	Semantic Search	
Applicable for	Fully / Semi Structured as well as Unstructured data	
Preferable with	Natural language interface alongside Advanced search filters	

Maithri Vm

Hybrid Search—Amalgamation of Sparse and Dense vector…

— Uniting meaning of data with metadata to leverage deeper context

13 min read · May 14

16

Alyx

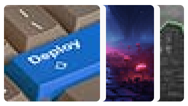
Semantic Search with FAISS

HuggingFace get_neareast_example and Cosine Similarity Search

9 min read · Jul 15

71 1

Lists

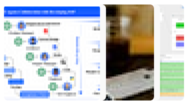


Predictive Modeling w/ Python

20 stories · 452 saves

Practical Guides to Machine Learning

10 stories · 519 saves

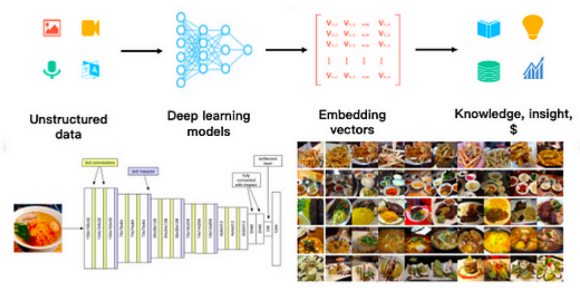


Natural Language Processing

669 stories · 283 saves

The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 133 saves



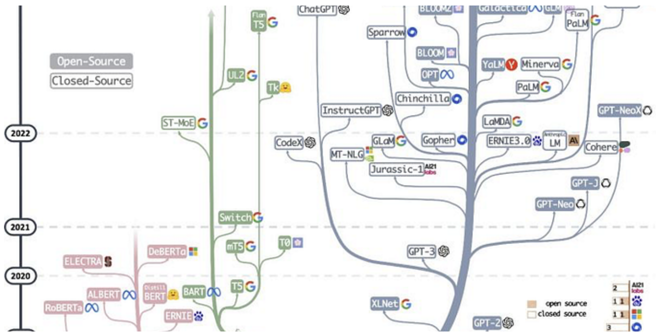
Jayita Bhattacharyya in GoPenAI

Primer on Vector Databases and Retrieval-Augmented Generation...

Vector Databases Generation (RAG) Langchain Pinecone HuggingFace Large...

9 min read · Aug 16

228 1



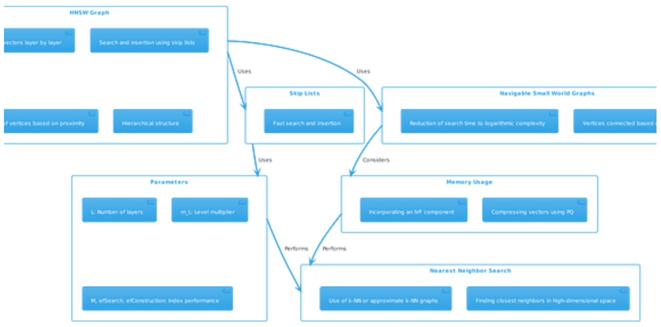
Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

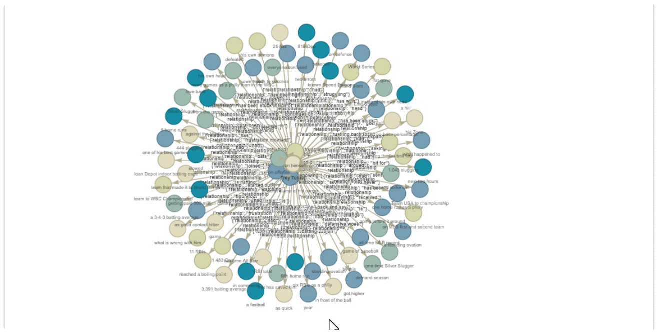
372



Bidhan R

Harnessing the Power of Hierarchical Navigable Small Wor...

Ever wondered how artificial intelligence systems retrieve precise, context-specific...



Wenqi Glantz in Better Programming

7 Query Strategies for Navigating Knowledge Graphs With...

Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies

3 min read · Jun 17

9

· 17 min read · 4 days ago

501 4

See more recommendations