The Journal of Open Source Software

SimilaritySearch.jl: Autotuned nearest neighbor indexes for Julia

Eric S. Tellez [1,2] **and Guillermo Ruiz** [1,3]

1 Consejo Nacional de Ciencia y Tecnología, México. **2** INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación, México. **3** CentroGEO Centro de Investigación en Ciencias de Información Geoespacial, México.

DOI: 10.21105/joss.04442

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @kose-y
- @ahwillia

Submitted: 23 May 2022
Published: 05 July 2022

Summary

This manuscript describes the MIT-licensed Julia (Bezanson et al., 2017) package `SimilaritySearch.jl` that provides algorithms to efficiently retrieve k nearest neighbors from a metric dataset and other related problems with no knowledge of the underlying algorithms, since our main structure, the `SearchGraph`, has autotuning capabilities. The package is designed to work in main memory and takes advantage of multithreading systems in most of its primary operations.

Statement of need

Similarity search algorithms are fundamental tools for many computer science and data analysis methods. For instance, they are among the underlying machinery behind efficient information retrieval systems (Luan et al., 2021; Witten et al., 1999), and they allow fast clustering analysis on large datasets (Jayaram Subramanya et al., 2019; Weng et al., 2021; Yu et al., 2020). Another example is how they can speed up the construction of all k nearest neighbor graphs, which are the input of non-linear dimensional reduction methods that are popularly used to visualize complex data (Amid & Warmuth, 2019; Lee & Verleysen, 2007; McInnes et al., 2018; Van der Maaten & Hinton, 2008). The number of potential applications is also increasing as the number of problems solved by deep learning methods proliferates, i.e., many deep learning internal representations are direct input for similarity search.

The k nearest neighbor problem

Given a metric dataset, $S \subseteq U$ and a metric distance function d, defined for any pair of elements in U, the k nearest neighbor search of q consists of finding the subset R that minimize $\sum_{u \in R} d(q,u)$ for all possible subsets of size k, i.e., $R \subset S$ and $|R| = k$. The problem can be solved easily with an exhaustive evaluation, but this solution is impractical when the number of expected queries is large or for high-dimensional datasets. When the dataset can be preprocessed, it is possible to overcome these difficulties by creating an *index*, i.e., a data structure to solve similarity queries efficiently. Depending on the dimensionality and size of the dataset, it could be necessary to trade speed for quality,[1] traditional methods leave this optimization to the user. Our approach has automated functions that simplify this task.

Our `SearchGraph` is based on the Navigable Small World (NSW) graph index (Y. A. Malkov & Yashunin, 2018) using a different search algorithm based on the well-known beam search meta-heuristic, smaller node degrees based on Spatial Access Trees (Navarro, 2002), and

[1] The quality is often measured as the `recall`, which is as a proportion of how many relevant results were found in a search; our package contains a function `macrorecall` that computes the average of this score for a set of query results.

autotuned capabilities. The details are provided in (Ruiz et al., 2015; Tellez et al., 2021; Tellez & Ruiz, 2022).

Alternatives

Y. Malkov et al. (2014) add a hierarchical structure to the NSW to create the Hierarchical NSW (HNSW) search structure. This index is a central component of the `hnswlib` and the `nmslib` libraries. Along with the HNSW, the `faiss` library also provides a broad set of efficient implementations of metric, hashing, and product quantization indexes. Dong et al. (2011) introduce the NN Descent method, which uses the graph of neighbors as index structure; it is the machinery behind `PyNNDescent`, which is behind the fast computation of UMAP non-linear low dimensional projection.[2] Guo et al. (2020) introduce the *SCANN* index for inner product-based metrics and Euclidean distance, available at the SCANN repository based on hashing.

Currently, there are some packages dedicated to nearest neighbor search, for instance, `NearestNeighbors.jl`, `Rayuela.jl`, `HNSW.jl`, and a wrapper for the FAISS library, `Faiss.jl`, among other efforts.

Main features of `SimilaritySearch`

The `SearchGraph` struct is an approximate method designed to trade effectively between speed and quality. It has an integrated autotuning feature that almost free the users of any setup and manual model selection. In a single pass, the incremental construction adjusts the index parameters to achieve the desired performance, optimizing both search speed and quality or a minimum quality. This search structure is described in Tellez & Ruiz (2022), which uses the `SimilaritySearch.jl` package as implementation (0.9 version series). Previous versions of the package are benchmarked in Tellez et al. (2021).

The main set of functions are:

- `search`: Solves a single query.
- `searchbatch`: Solves a set of queries.
- `allknn`: Computes the k nearest neighbors for all elements in an index.
- `closestpair`: Computes the closest pair in a metric dataset.
- `neardup`: Removes near-duplicates from a metric dataset.

Note that our implementations produce complete results with *exact* indexes and will produce approximate results when *approximate* indexes are used.

`SimilaritySearch.jl` can be used with any semi-metric, as defined in the package `Distances.jl`. Note that a number of distance functions for vectors, strings, and sets are also available in our package.

The complete set of functions and structures are detailed in the documentation.[3]

Installation

The package is available in the Julia's integrated package manager:

```
using Pkg
Pkg.add("SimilaritySearch")
```

[2] https://github.com/lmcinnes/umap.
[3] https://sadit.github.io/SimilaritySearch.jl/

A brief example and a comparison with alternatives

As an example, we used the set of 70k hand-written digits MNIST dataset (LeCun et al., 1998) (using the traditional partition scheme of 60k objects for indexing and 10k as queries). We use the `MLDatasets.jl` package for this matter (v0.6); each 28x28 image is loaded as a 784-dimensional vector using 32-bit floating-point numbers. We select the squared Euclidean distance as the metric.

```julia
 1  using SimilaritySearch, MLDatasets
 2
 3  function load_data()
 4    train, test = MNIST(split = :train), MNIST(split = :test)
 5    (w, h, n), m = size(train.features), size(test.features, 3)
 6    db = MatrixDatabase(reshape(train.features, w * h, n))
 7    queries = MatrixDatabase(reshape(test.features, w * h, m))
 8    db, queries
 9  end
10
11  function example(k, dist = SqL2Distance())
12    db, queries = load_data()
13    G = SearchGraph(; dist, db)
14    index!(G; parallel_block = 512)
15    id, dist = searchbatch(G, queries, k)
16    point1, point2, mindist = closestpair(G)
17    idAll, distAll = allknn(G, k)
18  end
19
20  example(32)
```

The function `example` loads the data (line 12), creates the index (line 14), and then finds all k nearest neighbors of the test in the indexed partition as a batch of queries (line 15). The same index is used to compute the closest pair of points in the train partition (line 16) and compute all k nearest neighbors on the train partition (line 17) for $k = 32$.

For this, we used an Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz workstation with 256 GiB RAM using GNU/Linux CentOS 8. Our system has 32 cores (64 threads), and we use all threads in all tested systems. For instance, we used `SimilaritySearch.jl` v0.9.3 and `Julia` 1.7.2. Table 1 compares the running times of `SearchGraph` (SG). We consider different autotuned versions calling `optimize!(G, MinRecall(r))` after the `index!` function call, for different expected recall scores, it defaults to `ParetoRecall`. We also compare with a parallel brute-force algorithm (replacing lines 13-14 with `ExhaustiveSearch(; dist, db)`).

Table 1: Performance comparison of running several similarity methods on the MNIST dataset. Smaller time costs and memory are desirable while high recall scores (close to 1) are better.

method	build	opt.	searchbatch	closestpair	allknn	mem.	allknn
	cost (s)	cost (s)	cost (s)	cost (s)	cost (s)	(MB)	recall
ExhaustiveSearch	0.0	0.0	3.56	22.18	21.65	179.44	1.00
SG ParetoRecall	0.91	0.0	0.10	0.29	0.41	182.22	0.78
SG `MinRecall(0.6)`	"	0.10	0.04	0.11	0.19	"	0.66
SG `MinRecall(0.9)`	"	0.12	0.13	0.46	0.61	"	0.86
SG `MinRecall(0.95)`	"	0.23	0.15	0.55	0.75	"	0.93
SCANN	25.11	-	-	-	2.14	201.95	1.00
HNSW (FAISS)	1.91	-	-	-	1.99	195.02	0.99
PyNNDescent	45.09	-	-	-	9.94	430.42	0.99

Comparison with alternatives

We also indexed and searched for all k nearest neighbors using the default values for the HNSW, PyNNDescent, and SCANN nearest neighbor search indexes. All these operations were computed using all available threads. Note that high recall scores indicate that the default parameters can be adjusted to improve search times; nonetheless, optimizing parameters also imply using a model selection procedure that requires more computational resources and knowledge about the packages and methods. Our `SearchGraph` (SG) method performs this procedure in a single pass and without extra effort by the user. Note that we run several optimizations that use the same index and spend a small amount of time effectively trading between quality and speed; this also works for larger and high-dimensional datasets as benchmarked in Tellez & Ruiz (2022). Finally, short-lived tasks like computing all k nearest neighbors for non-linear dimensional reductions (e.g., data visualization) also require low build costs; therefore, a complete model selection is prohibitive, especially for large datasets.

Final notes

`SimilaritySearch.jl` provides a metric-agnostic alternative for similarity search in high-dimensional datasets. Additionally, our autotuning feature is a milestone in the nearest neighbor community since it makes the technology more accessible for users without profound knowledge in the field. More examples and notebooks (Pluto and Jupyter) are available in the sibling repository https://github.com/sadit/SimilaritySearchDemos.

Acknowledgements

The authors would like to thank the reviewers and the editor for their valuable time; their suggestions improved the quality of this manuscript. This research used the computing infrastructure of the *Laboratorio de GeoInteligencia Territorial* at *CentroGEO Centro de Investigación en Ciencias de Información Geoespacial*, Aguascalientes, México.

References

Amid, E., & Warmuth, M. K. (2019). TriMap: Large-scale dimensionality reduction using triplets. *CoRR*, *abs/1910.00204*. http://arxiv.org/abs/1910.00204

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Dong, W., Moses, C., & Li, K. (2011). Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th International Conference on World Wide Web*, 577–586. https://doi.org/10.1145/1963405.1963487

Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., & Kumar, S. (2020). Accelerating large-scale inference with anisotropic vector quantization. *37th International Conference on Machine Learning, ICML 2020*, *PartF168147-5*, 3845–3854. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85105244400&partnerID=40&md5=e3ab797435367141112b5e5843b2cb1e

Jayaram Subramanya, S., Devvrit, F., Simhadri, H. V., Krishnawamy, R., & Kadekodi, R. (2019). Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems*, *32*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Lee, J. A., & Verleysen, M. (2007). *Nonlinear dimensionality reduction* (Vol. 1). Springer.

Luan, Y., Eisenstein, J., Toutanova, K., & Collins, M. (2021). Sparse, Dense, and Attentional Representations for Text Retrieval. *Transactions of the Association for Computational Linguistics*, *9*, 329–345. https://doi.org/10.1162/tacl_a_00369

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(4), 824–836. https://doi.org/10.1109/tpami.2018.2889473

Malkov, Y., Ponomarenko, A., Logvinov, A., & Krylov, V. (2014). Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, *45*, 61–68. https://doi.org/10.1016/j.is.2013.10.006

McInnes, L., Healy, J., & Melville, J. (2018). *UMAP: Uniform manifold approximation and projection for dimension reduction*. arXiv. https://doi.org/10.48550/ARXIV.1802.03426

Navarro, G. (2002). Searching in metric spaces by spatial approximation. *The VLDB Journal*, *11*(1), 28–46. https://doi.org/10.1007/s007780200060

Ruiz, G., Chávez, E., Graff, M., & Téllez, E. S. (2015). Finding near neighbors through local search. *International Conference on Similarity Search and Applications*, 103–109. https://doi.org/10.1007/978-3-319-25087-8_10

Tellez, E. S., & Ruiz, G. (2022). *Similarity search on neighbor's graphs with automatic pareto optimal performance and minimum expected quality setups based on hyperparameter optimization*. arXiv. https://doi.org/10.48550/ARXIV.2201.07917

Tellez, E. S., Ruiz, G., Chavez, E., & Graff, M. (2021). A scalable solution to the nearest neighbor search problem through local-search methods on neighbor graphs. *Pattern Analysis and Applications*, *24*(2), 763–777. https://doi.org/10.1007/s10044-020-00946-w

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(11).

Weng, S., Gou, J., & Fan, Z. (2021). h-DBSCAN: A simple fast DBSCAN algorithm for big data. In V. N. Balasubramanian & I. Tsang (Eds.), *Proceedings of the 13th asian conference on machine learning* (Vol. 157, pp. 81–96). PMLR. https://proceedings.mlr.press/v157/weng21a.html

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann.

Yu, Q., Chen, K.-H., & Chen, J.-J. (2020). Using a set of triangle inequalities to accelerate k-means clustering. *International Conference on Similarity Search and Applications*, 297–311. https://doi.org/10.1007/978-3-030-60936-8_23