НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
март–апрель 2025 Том 25 № 2 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
March–April 2025 Vol. 25 No 2 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

doi: 10.17586/2226-1494-2025-25-2-339-344

Vector search using method of clustering using ensemble of oblivious trees

Nikita A. Tomilov[1], Vladimir P. Turov[2]✉, Alexander A. Babayants[3], Alexey V. Platonov[4]

[1,2,3,4] ITMO University, Saint Petersburg, 197101, Russian Federation

[1] programmer174@icloud.com, https://orcid.org/0000-0001-9325-0356
[2] firemoon@icloud.com✉, https://orcid.org/0009-0009-1470-7633
[3] babayants.alexander@gmail.com, https://orcid.org/0009-0004-6367-6398
[4] avplatonov@itmo.ru, https://orcid.org/0000-0002-8485-1296

Abstract

Information retrieval using machine learning algorithms is based on transforming the original multimodal documents into vector representations. These vectors are then indexed, and the search is performed within this index. A popular method for indexing is vector clustering such as with *k*-nearest neighbors. We propose a clustering method based on an ensemble of Oblivious Decision Trees and introduce a vector search algorithm built on this method. The proposed clustering method is deterministic and supports parameter serialization for the ensemble. The essence of the method involves training an ensemble of binary or ternary Oblivious Trees. This ensemble is then used to compute a hash for each of the original vectors. Vectors with the same hash are considered to belong to the same cluster. For searching, several clusters are selected whose centroids are closest to the vector representation of the search query followed by a full search of the vector representations within the selected clusters. The proposed method demonstrates search quality comparable to widely used industry-standard vector search libraries, such as Faiss, Annoy, and HNSWlib. For datasets with an angular distance metric, the proposed search method achieves accuracy equal to or better than existing solutions. For datasets with a Euclidean distance metric, the search quality is on par with existing solutions. The developed method can be applied to improve search quality in the development of multimodal search systems. The ability to serialize enables clustering data on one computational node and transferring ensemble parameters to another, allowing the proposed algorithm to be utilized in distributed systems.

Keywords

vector representations, vector search, embeddings, oblivious decision tree

For citation: Tomilov N.A., Turov V.P., Babayants A.A., Platonov A.V. Vector search using method of clustering using ensemble of oblivious trees. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2025, vol. 25, no. 2, pp. 339–344. doi: 10.17586/2226-1494-2025-25-2-339-344

УДК 004.021

Векторный поиск методом кластеризации с помощью ансамбля небрежных решающих деревьев

Никита Андреевич Томилов[1], Владимир Павлович Туров[2]✉, Александр Амаякович Бабаянц[3], Алексей Владимирович Платонов[4]

[1,2,3,4] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация

[1] programmer174@icloud.com, https://orcid.org/0000-0001-9325-0356
[2] firemoon@icloud.com✉, https://orcid.org/0009-0009-1470-7633
[3] babayants.alexander@gmail.com, https://orcid.org/0009-0004-6367-6398
[4] avplatonov@itmo.ru, https://orcid.org/0000-0002-8485-1296

Аннотация

Введение. Информационный поиск с использованием алгоритмов машинного обучения основывается на преобразовании исходных мультимодальных документов в векторные представления, далее строится индекс векторных представлений и производится поиск внутри индекса. Популярным способом построения индекса

© Tomilov N.A., Turov V.P., Babayants A.A., Platonov A.V., 2025

является кластеризация векторных представлений, например с помощью *k*-ближайших соседей. В работе предложен метод кластеризации с помощью ансамбля небрежных решающих деревьев, а также алгоритм векторного поиска на основе этого метода. Разработанный метод кластеризации является детерминированным и предоставляет возможность сериализации параметров ансамбля. **Метод.** Сущность метода состоит в обучении ансамбля двоичных или троичных небрежных решающих деревьев. Этот ансамбль используется для вычисления хэша для каждого из исходных векторных представлений. Векторные представления, имеющие одинаковый хэш, считаются принадлежащими к одному кластеру. Для поиска выбирается несколько кластеров, центроиды которых наиболее приближены к векторному представлению поискового запроса, после чего производится полный перебор векторных представлений выбранных кластеров. **Основные результаты.** Представленный метод показывает качество поиска, сравнимое с широко используемыми в индустрии библиотеками векторного поиска Faiss, Annoy и HNSWlib. Для протестированного набора данных с евклидовой метрикой расстояния предложенный метод поиска медленнее, чем существующие решения, но для протестированного набора данных с угловой метрикой расстояния результат сравним или лучше. **Обсуждение.** Разработанный метод может быть применен для улучшения качества поиска при создании мультимодальных поисковых систем. Возможность сериализации позволяет кластеризовать данные на одном вычислительном узле и передавать параметры ансамбля на другой вычислительный узел, что дает возможность использовать предложенный алгоритм в распределенных системах.

Ключевые слова

векторные представления, векторный поиск, эмбеддинг, небрежное решающее дерево

Ссылка для цитирования: Томилов Н.А., Туров В.П., Бабаянц А.А., Платонов А.В. Векторный поиск методом кластеризации с помощью ансамбля небрежных решающих деревьев // Научно-технический вестник информационных технологий, механики и оптики. 2025. Т. 25, № 2. С. 339–344 (на англ. яз.). doi: 10.17586/2226-1494-2025-25-2-339-344

Introduction

One of the methods for organizing search in large databases of heterogeneous data and documents involves a machine learning-based approach. This approach transforms data into representations that reflect the semantic structure of both textual [1] and multimodal documents in the form of vector embeddings being sequences of floating-point numbers. In this framework, efficient search can be achieved by constructing an index of these embeddings, transforming the search query into a vector embedding, and performing a nearest neighbor search using the constructed index [2]. Clustering, or grouping, vector embeddings is among the most popular indexing methods. In this method, the search index consists of cluster centroids. To perform a search, the system selects one or more centroids closest to the query and then performs an exhaustive search among the vector embeddings within the corresponding clusters. This indexing approach reduces the number of vector embeddings considered during the search to those within a single cluster [3]. A common clustering method for this purpose is the *k*-Nearest Neighbor (*k*NN) algorithm [4].

The most significant drawback of this method is its computational complexity. To determine which cluster a given vector embedding belongs to, it is necessary to calculate the distance from the embedding to all centroids of all existing clusters. It might also be necessary to, after assigning the embedding to the nearest cluster, recalculate the centroid of that cluster. Performing these computations requires not only access to the centroids of all clusters but also access to all vectors associated with the target centroid, which complicates the use of this algorithm in distributed systems. Less critical drawbacks include the use of a pseudorandom number generator during the clustering process, which may lead to different outputs for identical input data, as well as the limited number of hyper parameters available in the algorithm.

In this paper we propose a clustering method using an ensemble of Oblivious Decision Trees (ODT) [5] as well as an approximate search method leveraging this clustering. This clustering assigns a hash to each vector embedding which serves as a cluster identifier and can be used to implement a Bloom filter for vector embeddings [6]. The rules for computing this hash are deterministic and independent of the vector data, allowing them to be serialized and used separately from the vector data. Our hypothesis is that this clustering method overcomes the drawbacks of the *k*NN method while providing comparable search accuracy.

Description of proposed clustering method

In this study, we propose an approximate vector search method based on clustering using an ensemble of binary or ternary ODT. Since training trees require significant computational effort, we build the ensemble by randomly selecting a training sample of size $N = N_{dataset} \times TrainRatio$, where $N_{dataset}$ is the amount of vectors in the dataset, $0.1 \le TrainRatio \le 1$. We transform each vector embedding of dimension d from the sample into M vector embeddings of a smaller dimension d_m, created by randomly selecting components from the original embedding. We also store a mapping table that links the indices of the components in the generated embeddings to their indices in the original embedding. This approach allows some components of the original embedding to remain unused or to appear multiple times, including multiple occurrences within the same generated embedding. After this transformation, we train a tree for each of the M sets of generated embeddings. These M trained trees form the ensemble.

The essence of the algorithm for constructing a ternary ODT of *depth* for vector embeddings of dimension d lies in selecting a set of elements $(K_j; X_j)$, where $1 \le j \le depth$, $1 \le K_j \le d_m$, at each tree level, such that the set of vector embeddings V is divided into three non-overlapping

subsets based on whether the K_j-th component of a vector embedding from V is less than X_j-*delta*, greater than X_j + *delta*, or approximately equal to X_j within the *delta* tolerance. This *delta* parameter defines the size of a group and is a configurable parameter of the structure. The partitioning rule is a hyper parameter, and in this study, we use a partitioning approach that minimizes the variance of distances between each vector embedding and the average vector embedding for that group.

The process is iterative and continues until the required depth is reached. At each step, the next division is performed for each of the three groups formed by the previous division. Thus, after the first division, the original set of vector representations V is split into three parts, V_1, V_2, and V_3. On the next stage (the second level), each of the groups V_1, V_2, and V_3 is divided into three parts again, resulting in a total of nine groups, and so on. A distinctive feature of ODT is that the division for all groups at level j is done using the same value X_j [7]. Consequently, when dividing V, a situation may arise where the K_j-th component of all vector representations in V is strictly greater than or strictly less than X_j and thus all vector representations end up in one resulting group V_1, while the groups V_2, and V_3 are empty. This is a normal situation, but as a result, the final number of non-empty groups may be less than $\log_3(depth)$.

After performing the calculations, the tree is described by an array of tuples $(K_j; X_j)$ and the *delta* parameter. Each node at depth j represents the operation "is the K_j-th value of the vector representation less than X_j-*delta*, greater than X_j + *delta*, or approximately equal to X_j within the tolerance of *delta*".

The process of individual tree construction is repeated for each of the M sets of vector representations to form the ensemble. After constructing the ensemble of trees, at the final stage, it is necessary to use the previously saved correspondence table to restore the K_j index for each element, associating the index of the element to be compared, within the M subvector with the d_m, dimensions, with the index of this element in the original vector representation of size d.

For each vector from the original dataset, calculations can be performed using the obtained tree, encoding the comparison result at each tree level j as '0' if the K_j-th component of the vector is less than X_j-*delta*, '1' if it is greater than X_j + *delta*, or 'E' in the case of approximate equality. This results in a ternary string of length *depth* which serves as the hash of the vector, and the length of this hash is equal to d_m. When creating an ensemble of M trees, M hashes of the length *nBits* are generated. We can call these individual hashes *SH*. These individual hashes can be combined into a single hash called *TH* of length M *nBits*. This is illustrated in Fig. 1. The exact order in which the combination occurs can be arbitrary, but it must be the same for each vector representation across the current set of vector representations.

In other words, by using M ODT, we can assign each vector from the original dataset a hash string *TH*. With this approach, similar vectors will have the same hash *TH*, meaning that vectors from the original dataset with identical hashes will belong to the same cluster.

A binary tree is a special case of a ternary tree where the value of delta equals zero. Consequently, during construction, each tree node splits into two parts: "greater" and "less than or equal". The resulting hash *TH* is then a binary string containing '0' and '1', thus it can be converted to an integer number.

Thanks to the fact that the tree is described by an array of tuples $(K_j; X_j)$, the tree can be serialized and transmitted separately from the data it was trained on. This allows, for example, the hash of a vector to be computed directly in client-side code when organizing client-server interactions, even without access to the data stored on the server, or in distributed systems without access to the data stored

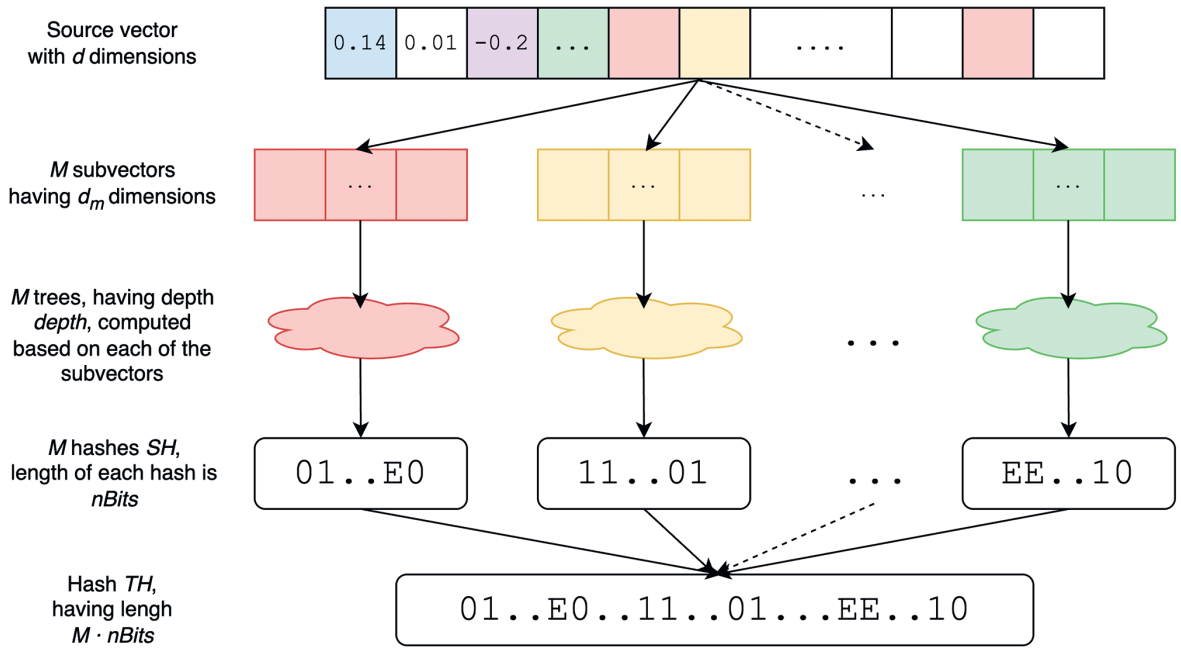

Fig. 1. The process of transforming the original vector to its hash

on another cluster node. In addition, the large number of parameters and hyper parameters for selecting the appropriate partition allows fine-tuning of the ensemble for specific tasks. However, a significant drawback is the computational complexity of building the tree ensemble due to the exhaustive search of X_j values at each depth level. Even without considering the computation of distance variance, the number of iterations required for the search has a quadratic dependency on the number of input data, with additional increasing factors from the clustering parameters. Nevertheless, thanks to good parallelization, this drawback can be mitigated by using parallel or even heterogeneous computing (e.g., employing both CPU and GPU) [8].

In order to perform a search on data clustered using ODT, after clustering by hash value, it is necessary to compute and store the average vector for each cluster.

Description of proposed search method

When performing the *k*NN search for the vector embedding V_q obtained from the search query, we select N_q clusters, whose centroids are closest to V_q, from all clusters, where $N_q = \log_{10}N$, with N being the total number of clusters. Additionally, if present, we select the cluster whose hash matches the hash of V_q. Next, for each vector embedding in the selected clusters, we compute its distance to V_q. Then, for each cluster, we select k vector embeddings that are closest to V_q. The resulting vector embeddings are sorted in ascending order of distance to the k nearest embeddings, and the final result consists of the k embeddings closest to V_q.

The advantages of the algorithm include its versatility. The algorithm is independent of the distance metric and works with any method of storing vector embeddings, as long as the method allows storing metadata for each vector and/or supports grouping vector embeddings. However, the proposed algorithm does not completely eliminate the exhaustive search of vector embeddings within a group, which negatively impacts search time, especially in cases where parallel computations cannot be utilized.

With the exception of the step involving the selection of the cluster whose hash matches the hash of the query vector embedding, the proposed search algorithm is independent of the clustering algorithm. This allows for comparison of different clustering algorithms, as will be demonstrated in the comparison of the proposed clustering method using an ensemble of ODT with clustering using the *k*NN method.

Experiment setup

To validate the proposed vector search algorithm, we implemented it in the Kotlin programming language. For storage, we chose a previously implemented library for storing vector embeddings that group the embeddings into a single page file.

To test the proposed solution, we selected two datasets containing different vectors: the NYT-256-angular[1] and Fashion-MNIST-784-euclidean [9] datasets taken from the popular ANN-Benchmarks [10] collection which serves as the industry standard for measuring the performance of approximate vector search algorithms. Specifically, Fashion-MNIST contains 50,000 vector embeddings of dimension 784, representing 32 × 32 pixel images where each component of the vector encodes the brightness of the corresponding pixel ranging from 0 to 255. The NYT dataset includes 290,000 vector embeddings of text articles from the New York Times, each containing 256 components. These two datasets differ in the semantics of the encoded data, value ranges, and distance metrics, allowing us to evaluate the algorithm performance on different types of input data. However, because the tree training implementation is time-intensive, we selected only the first 5,000 vector embeddings from each dataset to limit the tree training time to 12 hours.

The selected vector embeddings were transformed into file sets corresponding to the storage configurations listed below. The specific parameter values were chosen after numerous experiments, as they offer a balance between the algorithm accuracy, search speed, and the tree ensemble construction time.

As an alternative solution for comparison, we selected the traditional nearest neighbor clustering method which was previously implemented in the study on vector index compression [11]. In total, for the experiment, the original datasets were clustered and stored as follows:

1. Clustering vector embeddings using ternary ODT ensemble with the following parameters:
   — *delta* — 0.15 and 0.35;
   — M — 2, 4, 8;
   — d_m – 392 and 784 for the Fashion-MNIST dataset, 128 and 256 for the NYT dataset (in other words, 50 % and 100 % from the source dataset dimensionality);
   — *depth* — 2, 4, 8;
   — *TrainRatio* = 1.0 (in other words, all of the 5 thousand vectors were used to train the trees);
   — The partitioning rule: partition to minimize the variance of distances between each vector embedding and the average vector embedding for that group.
2. Clustering vector embeddings using ensemble of binary ODT, having the same parameters as ternary trees, except for *delta* which is 0;
3. Clustering vector embeddings using *k*NN clustering, maximum number of clusters being an arbitrary number between 15 and 15,000.

After obtaining the sets of clusters, we measured the search speed and accuracy using the following cluster variants:

— Clusters obtained by ternary tree ensembles for all parameter combinations.
— Clusters obtained by binary tree ensembles for all parameter combinations.
— Clusters obtained using the nearest neighbor method.

In addition to testing the performance of the proposed solution and the proposed benchmark, we also conducted performance testing of libraries that are industry standards

[1] Newman D. Bag of Words // UCI Machine Learning Repository. 2008. doi: 10.24432/C5ZG6P

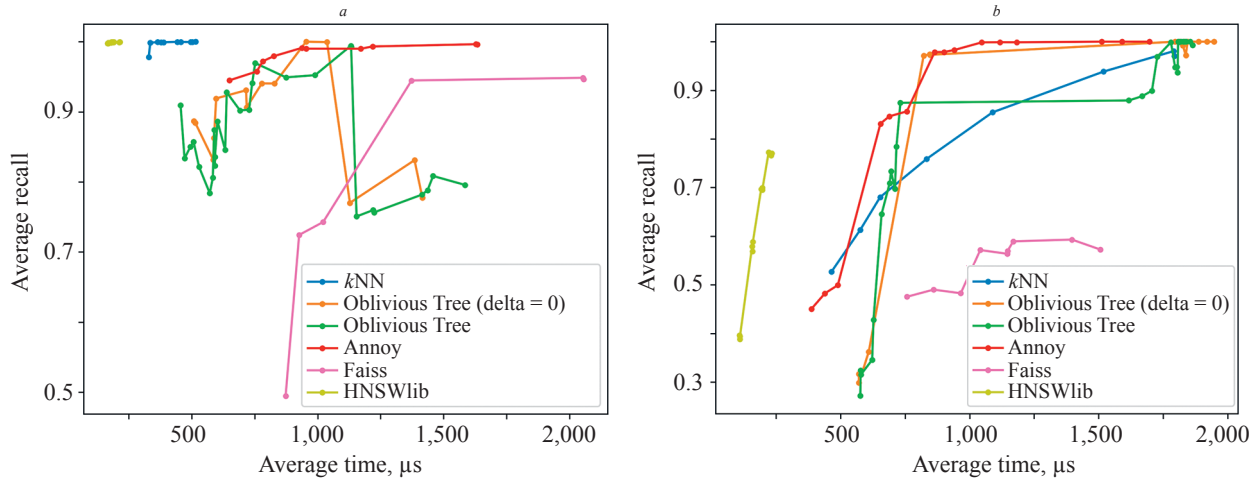

Fig. 2. Comparison results between *k*NN clustering and clustering using decision trees: Fashion-MNIST dataset (*a*); NYT dataset (*b*)

for approximate vector search, namely Annoy[1], Faiss [12], and HNSWLib [13]. The parameters of these storage systems were chosen to match those in ANN-Benchmarks [10], and the same 5,000 vector embeddings were selected for indexing. The test setup has the following specifications: AMD Ryzen 7 7700X 8C16T; 64GB RAM; NVMe WD SN850X 2TB; OS Ubuntu 22.04; OpenJDK 17; a tool for comparing vector search algorithms, developed in previous research [14], using Java Microbenchmark Harness [15].

Experiment results

The measurement results are presented in Fig. 2 where the best storage system is the one with the higher response and lower search time.

For the Fashion-MNIST dataset [9], the proposed clustering solution using trees cannot compete with *k*NN clustering under any combination of tree construction parameters. Since the vector embeddings in this dataset represent images of ten types of clothing, such data clusters well into a small number of clusters, no more than 1,000. Exhaustive search across these clusters provides high accuracy, and with such a small number of clusters and a small volume of data within each cluster, the search is fast. In contrast, clustering using ODT, with different input parameter sets, produces between 16 and 20,000 clusters. When there are too few clusters, leading to a large number of vector embeddings within each cluster, the search within the cluster becomes too slow. On the other hand, with too many clusters, the number of clusters to be searched becomes too small, resulting in lower accuracy. Acceptable accuracy is only achieved for parameter sets that produce a clustering with fewer than 1,000 clusters, but with worse data grouping, leading to lower search accuracy.

For the NYT dataset, the proposed clustering solution using trees can compete with *k*NN clustering for certain parameter combinations. Clustering with trees still produces a large number of clusters ranging from 15 to 246,000; however, in this case, clustering also generates up to 15,000 clusters, which reduces the speed advantage caused by a small number of clusters. This increase compared to the previous dataset is due to the nature of the data which consists of dense embeddings of text documents. For parameter sets producing up to 100 clusters, clustering with trees provides a higher response compared to *k*NN clustering, at the cost of a slight increase in search time.

For both datasets, comparison with industrial standards for vector search shows that the proposed search method can compete in terms of both speed and accuracy with Annoy and Faiss, and for the NYT dataset, it can compete in accuracy with HNSW at the cost of longer search time.

Conclusion

For the selected datasets, the proposed search method in clusters generated by Oblivious Decision Trees (ODT) achieves search quality comparable to or better than that of the *k*NN method or libraries like Faiss, Annoy, and HNSW. However, for the Fashion-MNIST dataset, clustering using ODT demonstrates significantly slower search performance at the same quality metric values.

These results suggest that the proposed clustering and search method is comparable in search quality to industry standards. This makes further exploration of ODT for vector search worthwhile, specifically in accelerating the ensemble construction process and reducing the number of trees in the ensemble. The former would enable faster index construction, while the latter would speed up the search itself.

[1] GitHub — spotify/annoy: Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk. Available at: https://github.com/spotify/annoy, unrestricted. Language: English. (accessed: 14.06.2024).

References

1. Grbovic M., Cheng H. Real-time personalization using embeddings for search ranking at Airbnb. *Proc. of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 311–320. https://doi.org/10.1145/3219819.3219885
2. Berry M.W., Drmac Z., Jessup E.R. Matrices, vector spaces, and information retrieval. *SIAM Review*, 1999, vol. 41, no. 2, pp. 335–362. https://doi.org/10.1137/S0036144598347035
3. Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z Fast nearest neighbor search in medical image databases. *Proc. of the 22th International Conference on Very Large Data Bases (VLDB '96)*, 1996, pp. 215–226. https://doi.org/10.5555/645922.673493
4. Seidl T., Kriegel H.-P. Optimal multi-step k-nearest neighbor search. *ACM SIGMOD Record*, 1998, vol. 27, no. 2, pp. 154–165. https://doi.org/10.1145/276305.276319
5. Lou Y., Obukhov M. BDT: Gradient Boosted Decision Tables for high accuracy and scoring efficiency. *Proc. of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*, 2017, pp. 1893–1901. https://doi.org/10.1145/3097983.3098175
6. Chakrabarti S. Efficient image retrieval using multi neural hash codes and bloom filters. *Proc. of the IEEE International Conference for Innovation in Technology (INOCON)*, 2020, pp. 1–6. https://doi.org/10.1109/INOCON50539.2020.9298228
7. Rokach L., Maimon O. *Data Mining with Decision Trees: Theory and Applications*. 2nd ed. World Scientific Publishing, 2014. 328 p.
8. Fumero J., Papadimitriou M., Zakkak F.S., Xekalaki M., Clarkson J., Kotselidis C. Dynamic application reconfiguration on heterogeneous hardware. *Proc. of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2019, pp. 165–178. https://doi.org/10.1145/3313808.3313819
9. Xiao H., Rasul K., Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, 2017, arXiv:1708.07747. https://doi.org/10.48550/arXiv.1708.07747
10. Aumüller M., Bernhardsson E., Faithfull A. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Lecture Notes in Computer Science*, 2017, vol. 10609, pp. 34–49. https://doi.org/10.1007/978-3-319-68474-1_3
11. Tomilov N.A., Turov V.P., Babayants A.A., Platonov A.V. A method of storing vector data in compressed form using clustering. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2024, vol. 24, no. 1, pp. 112–117. (in Russian). https://doi.org/10.17586/2226-1494-2024-24-1-112-117
12. Douze M., Guzhva A., Deng C., Johnson J., Szilvasy G., Mazaré P.-E., Lomeli M., Hosseini L., Jégou H. The Faiss library. *arXiv*, 2024, arXiv.2401.08281. https://doi.org/10.48550/arXiv.2401.08281
13. Malkov Y., Yashunin D. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020, vol. 42, no. 4. pp. 824–836. https://doi.org/10.1109/TPAMI.2018.2889473
14. Tomilov N.A., Turov V.P., Babayants A.A. Developing a tool to compare vector search algorithms. *Proc. of the 11th Congress of Young Scientists*, 2022, vol. 1. pp. 446–450. (in Russian).
15. Laaber C., Leitner P. An evaluation of open-source software microbenchmark suites for continuous performance assessment. *Proc. of the 15th International Conference on Mining Software Repositories (MSR '18)*, 2018, pp. 119–130. https://doi.org/10.1145/3196398.3196407

Литература

1. Grbovic M., Cheng H. Real-time personalization using embeddings for search ranking at airbnb // Proc. of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018. P. 311–320. https://doi.org/10.1145/3219819.3219885
2. Berry M.W., Drmac Z., Jessup E.R. Matrices, vector spaces, and information retrieval // SIAM Review. 1999. V. 41. N 2. P. 335–362. https://doi.org/10.1137/S0036144598347035
3. Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z Fast nearest neighbor search in medical image databases // Proc. of the 22th International Conference on Very Large Data Bases (VLDB '96). 1996. P. 215–226. https://doi.org/10.5555/645922.673493
4. Seidl T., Kriegel H.-P. Optimal multi-step k-nearest neighbor search // ACM SIGMOD Record. 1998. V. 27. N 2. P. 154–165. https://doi.org/10.1145/276305.276319
5. Lou Y., Obukhov M. BDT: Gradient Boosted Decision Tables for high accuracy and scoring efficiency // Proc. of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17). 2017. P. 1893–1901. https://doi.org/10.1145/3097983.3098175
6. Chakrabarti S. Efficient image retrieval using multi neural hash codes and bloom filters // Proc. of the IEEE International Conference for Innovation in Technology (INOCON). 2020. P. 1–6. https://doi.org/10.1109/INOCON50539.2020.9298228
7. Rokach L., Maimon O. Data Mining with Decision Trees: Theory and Applications / 2nd ed. World Scientific Publishing, 2014. 328 p.
8. Fumero J., Papadimitriou M., Zakkak F.S., Xekalaki M., Clarkson J., Kotselidis C. Dynamic application reconfiguration on heterogeneous hardware // Proc. of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 2019. P. 165–178. https://doi.org/10.1145/3313808.3313819
9. Xiao H., Rasul K., Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms // arXiv. 2017. arXiv:1708.07747. https://doi.org/10.48550/arXiv.1708.07747
10. Aumüller M., Bernhardsson E., Faithfull A. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms // Lecture Notes in Computer Science. 2017. V. 10609. P. 34–49. https://doi.org/10.1007/978-3-319-68474-1_3
11. Томилов Н.А., Туров В.П., Бабаянц А.А., Платонов А.В. Метод хранения векторных представлений в сжатом виде с применением кластеризации // Научно-технический вестник информационных технологий, механики и оптики. 2024. Т. 24, № 1. С. 112–117. https://doi.org/10.17586/2226-1494-2024-24-1-112-117
12. Douze M., Guzhva A., Deng C., Johnson J., Szilvasy G., Mazaré P.-E., Lomeli M., Hosseini L., Jégou H. The Faiss library // arXiv. 2024. arXiv.2401.08281. https://doi.org/10.48550/arXiv.2401.08281
13. Malkov Y., Yashunin D. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020. V. 42. N 4. P. 824–836. https://doi.org/10.1109/TPAMI.2018.2889473
14. Туров В.П., Томилов Н.А., Бабаянц А.А. Разработка инструмента сравнения алгоритмов векторного поиска // XI Конгресс молодых учёных: Сборник научных трудов. СПб.: Национальный исследовательский университет ИТМО, 2022. Т. 1. С. 446–450.
15. Laaber C., Leitner P. An evaluation of open-source software microbenchmark suites for continuous performance assessment // Proc. of the 15th International Conference on Mining Software Repositories (MSR '18). 2018. P. 119–130. https://doi.org/10.1145/3196398.3196407

Authors

Nikita A. Tomilov — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 57225127284, https://orcid.org/0000-0001-9325-0356, programmer174@icloud.com

Vladimir P. Turov — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 58910796700, https://orcid.org/0009-0009-1470-7633, firemoon@icloud.com

Alexander A. Babayants — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 58910002300, https://orcid.org/0009-0004-6367-6398, babayants.alexander@gmail.com

Alexey V. Platonov — PhD, Associate Professor, Associate Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 57197736275, https://orcid.org/0000-0002-8485-1296, avplatonov@itmo.ru

Авторы

Томилов Никита Андреевич — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 57225127284, https://orcid.org/0000-0001-9325-0356, programmer174@icloud.com

Туров Владимир Павлович — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 58910796700, https://orcid.org/0009-0009-1470-7633, firemoon@icloud.com

Бабаянц Александр Амаякович — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 58910002300, https://orcid.org/0009-0004-6367-6398, babayants.alexander@gmail.com

Платонов Алексей Владимирович — кандидат технических наук, доцент, доцент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 57197736275, https://orcid.org/0000-0002-8485-1296, avplatonov@itmo.ru

Received 08.11.2024
Approved after reviewing 23.01.2025
Accepted 19.03.2025

Статья поступила в редакцию 08.11.2024
Одобрена после рецензирования 23.01.2025
Принята к печати 19.03.2025