UDC 004.424.4
DOI: 10.20535/SRIT.2308-8893.2023.4.05

SEMI-SUPERVISED INVERTED FILE INDEX APPROACH FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

A. BAZDYREV

Abstract. This paper introduces a novel modification to the Inverted File (IVF) index approach for approximate nearest neighbor search, incorporating supervised learning techniques to enhance the efficacy of intermediate clustering and achieve more balanced cluster sizes. The proposed method involves creating clusters using a neural network by solving a task to classify query vectors into the same bucket as their corresponding nearest neighbor vectors in the original dataset. When combined with minimizing the standard deviation of the bucket sizes, the indexing process becomes more efficient and accurate during the approximate nearest neighbor search. Through empirical evaluation on a test dataset, we demonstrate that the proposed semi-supervised IVF index approach outperforms the industry-standard IVF implementation with fixed parameters, including the total number of clusters and the number of clusters allocated to queries. This novel approach has promising implications for enhancing nearest-neighbor search efficiency in high-dimensional datasets across various applications, including information retrieval, natural language search, recommendation systems, etc.

Keywords: approximate nearest neighbor search, inverted file index, high-dimensional data, machine learning.

INTRODUCTION

Approximate Nearest Neighbor (ANN) [1] search is a fundamental problem in many data-driven applications, spanning domains such as information retrieval, image processing, natural language search, and recommendation systems. The efficient retrieval of similar data points from vast datasets is critical for tasks that involve high-dimensional data representations, where exhaustive search methods become computationally infeasible. As the dataset size grows, the computational cost of performing an exact nearest neighbor search using brute force algorithms becomes prohibitive. Brute force approaches involve comparing each query vector with every data point in the dataset, leading to computational inefficiencies and impractical execution times for large datasets. Approximate nearest neighbor algorithms offer a trade-off between search accuracy and efficiency, allowing for the retrieval of reasonably accurate results within a significantly reduced search space. By intelligently approximating the nearest neighbors, these algorithms enable faster exploration of large datasets, making them essential for real-world applications where timely responses are crucial, such as image and text search, recommendation systems, and similarity-based clustering.

One popular approach in ANN is the Inverted File (IVF) index method [2]. Originally, the IVF index was an inverted indexing technique that partitions the dataset into a set of Voronoi cells or “buckets” [3]. Each bucket corresponds to a cluster of data points, and the indices of data points within each bucket are stored efficiently. During the search process, queries are mapped to their corresponding buckets, and the search is constrained to the nearest neighbors within these buckets, significantly reducing the search space and accelerating the process.

The standard IVF index has shown remarkable performance gains in nearest neighbor search tasks. However, it faces challenges in scenarios with unevenly distributed data, leading to imbalanced bucket sizes [4]. These imbalances can result in a suboptimal trade-off between search efficiency and accuracy, as some buckets might be excessively populated, while others remain underutilized. In addition to challenges posed by unevenly distributed data and imbalanced bucket sizes, another significant issue that the standard IVF index may encounter relates to the formation of centroid clusters. The standard approach typically relies on unsupervised clustering techniques to create the centroids or representatives for each bucket. This process can potentially lead to suboptimal cluster assignments, especially when the training data for centroid formation is insufficient or poorly representative of the underlying data distribution.

To address this limitation, we propose a novel modification to the IVF index method that leverages supervised learning techniques. Specifically, we train classification neural networks to assign query vectors to their most appropriate bucket, based on the similarity to vectors in the dataset. Moreover, we incorporate an optimization objective to minimize the standard deviation of the bucket sizes, further refining the indexing process. By doing so, we aim to achieve more balanced cluster sizes, effectively mitigating the impact of unevenly distributed data.

PRELIMINARIES

Let's formulate a general ANN problem. Let $X = \{x_i \in \mathbb{R}^d \mid i = \overline{1, N}\}$ be a set of N d-dimensional vectors representing the data points in the dataset. The objective of ANN search is to efficiently find, for a given query vector $q \in \mathbb{R}^d$, an approximate nearest neighbor $x^* \in X$ such that the distance between q and x^* is minimized.

In the Inverted File Index (IVF) approach, we partition the dataset X into K disjoint subsets or buckets, denoted as $B_1, B_2 \ldots B_K$. Each bucket corresponds to a subset (cluster) of vectors in X with corresponding centroids c_i — centroid of corresponding B_i.

The ANN search with the IVF index can be formulated as follows. Given the metric function *dist*, a query vector $q \in \mathbb{R}^d$, the goal is to find the bucket B_{query}, with a corresponding centroid c_{query} that minimizes the distance to the query vector — equation:

$$c_{query} = \underset{\{c_1 .. c_k\}}{\operatorname{argmin}} \, (dist\,(q, c_i)) \, .$$

Once the bucket B_{query}, is identified, we need to find x^* — approximate nearest neighbor within that bucket using brute force search — equation:

$$x^* = \underset{x \in B_{query}}{\operatorname{argmin}}(dist(q, x)) \, .$$

Optionally, to improve accuracy, it is possible to use several B_j adjoining to B_{query} buckets on the last step depending on the method hyperparameter set.

SEMI-SUPERVISED INVERTED FILE INDEX APPROACH

Let *dist* — some metric function (euclidian, manhattan, etc.).

Let $X = \{x_i \in \mathbb{R}^d \mid i = \overline{1,N}\}$ vectors representing the data points in the dataset.

Let $Q = \{q_i \in \mathbb{R}^d \mid i = \overline{1,M}\}$ — a set of M d-dimensional vectors with a similar distribution to real-life production queries be a queries training set, $M << N$.

Let $R = \{r_i \in X \mid r_i = \underset{x \in X}{\operatorname{argmin}}(dist(q_i, x)),\ i = \overline{1,M}\}$ — set of ground truth nearest neighbors (responses) from X for each $q_i \in Q$.

Let $K \in \mathbb{N}$ — method hyperparameter, a desired amount of buckets $B_1, B_2, \ldots, B_K$, such that $X = \bigcup_{i=1}^{K} B_i$ and $B_i \cap B_j = \varnothing$ if $i \neq j$.

Let $NN : \mathbb{R}^d \to \mathbb{R}^K$ – some vector function — equation:

$$NN_j(q_i) = P\{q_i \in B_j / r_i \in B_j\} \text{ for } j = \overline{1,K}, \tag{1}$$

where $P\{q_i \in B_j / r_i \in B_j\}$ — is a conditional probability that $q_i \in B_j$ given $r_i \in B_j$. In our case a multi-layer perceptron [5] with a final softmax layer — equation $\operatorname{softmax}_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ for $i = \overline{1,K}$, that distributes query vectors q_i into buckets $B_1, B_2, \ldots, B_K$. We also want this function to have a specific property, that it distributes query vectors $q_i \in Q$ to the same bucket as their corresponding responses $r_i \in R$.

We can estimate the NN's parameters using the maximum likelihood estimation method [6; 7], if we consider the task as a standard softmax multiclass classification with a cross-entropy loss function — equation $CE(y, \hat{y}) = -\sum_{i=1}^{K} y_i \log(\tilde{y}_i)$. If we consider Q as an input training set and on each epoch step we can calculate actual training targets Y as follows $Y = \{\underset{j=1,K}{\arg\max}(\{NN_j(r_i)\}),\ i = \overline{1,M}\}$ — for each training query we assign its ground truth nearest neighbor's bucket as a target bucket. As a result of NN training, we can explicitly distribute input queries by buckets — equation $bucket(q) = \underset{\boldsymbol{i}=1,\boldsymbol{K}}{\operatorname{argmax}}(\{NN_i(q)\})$ for $q \in \mathbb{R}^d$ and implicitly get the desired buckets $B_1, B_2, \ldots, B_K$ — equation:

$$B_j = \left\{ x \in X \mid \underset{i=1,K}{\arg\max}(\{NN_i(x)\}) = j \right\} \text{ for } j = \overline{1,K}. \tag{2}$$

STANDARD DEVIATION-BASED BUCKET SIZE REGULARIZATION

The vanilla approach proposed in the previous paragraph can produce imbalanced buckets $B_1, B_2, \ldots, B_K$ in the result, for example, *NN* will distribute all the query items in the single bucket, so there will be no full power use of the IVF index. If we want the most efficient computational power of the IVF index method, then we obviously need buckets of the most equal size so that the expectation of the search time of a brute force search over a random bucket takes the minimum time. Let $S = \{s_i = |B_i| \mid i = \overline{1, K}\}$ — set of buckets sizes after we have trained NN that distributes query vectors by buckets. We can calculate the standard deviation of the dataset *S*: $\sigma(S) = \sqrt{\left(\frac{\sum (s_i - \bar{s})^2}{N - 1}\right)}$. If we want to have buckets of approximately equal sizes then we need to minimize $\sigma(S)$. The problem here is that this function is not differentiable with respect to the parameters of the *NN* model, so we need to use a differentiable approximation of $\sigma(S)$.

Using equations (1), (2) we can calculate the expectation of size for each bucket as follows — equation:

$$\widetilde{s_j} = \sum_{i=1}^{N} NN_j(x_i) \text{ for } x_i \in X; \text{ for } j = \overline{1, K}. \tag{3}$$

So, we can have $\widetilde{S} = \{s_i \mid j = \overline{1, K}\}$ — set of expectations of bucket sizes after we have trained *NN* that distributes query vectors by buckets. And $\sigma(\widetilde{S})$ which is differentiable with respect to the parameters of the *NN* model.

Finally, we can introduce a combined multiclass cross-entropy loss function with std-based bucket size regularization in equation:

$$L(y, \hat{y}, X) = \left(-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} \log(\widetilde{y}_{ij})\right) + \gamma * \sigma(\widetilde{S}), \tag{4}$$

where $\left(-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} \log(\widetilde{y}_{ij})\right)$ is a standard cross-entropy component; $\sigma(\widetilde{S})$ — approximated standard deviation of bucket sizes and $\gamma \in [0, +\infty)$ — regularization scale.

TRAINING ALGORITHM

1. Defining *K* — desired number of buckets and *M* — desired maximum bucket size.
2. Initialization of multiclass classification *NN* weights [8].
3. On each training epoch:
   1. Calculate current epoch targets $Y = \{\underset{j=1,K}{\operatorname{argmax}}(\{NN_j(r_i)\})\}$.
   2. Calculate the multiclass cross-entropy loss component using $q_i \in Q$ as inputs and $y_i \in Y$ as targets.
   3. Calculate expectations of sizes for each cluster — equation (3).
   4. Calculate $\sigma(\widetilde{S})$ — std-regularization component.
   5. Calculate aggregated loss equation (4).
   6. Do the backpropagation step using stochastic gradient descent modification, for example, Adam [9], and update NN's weights.

4. After the training process is complete, we select the best checkpoint based on the desired performance metric, for example, precision where the actual maximum bucket size < *M*. If there is no such checkpoint in which the maximum actual bucket size is lower than the desired one, then select the checkpoint with the size closest to the desired one and display the corresponding warning.

It could also be useful to apply some dynamic scaling of γ regularization parameter to achieve better precision performance results.

EXPERIMENTAL RESULTS

We've used 3 different configurations in our experiments:

1. Both indexed and query data have a Normal distribution: $X \sim N(0,1)$; $Q \sim N(0,1)$.

2. Both indexed and query data have a skewed Exponential distribution: $X \sim Exponential\,(1)$; $Q \sim Exponential\,(1)$.

3. Indexed data has a Normal distribution and query data has an Exponential distribution that can be similar to different life scenarios: $X \sim N(0,1)$; $Q \sim Exponential(1)$.

In all cases we use 64-dimensional vectors. We also split query data Q to training and testing parts equally in order to minimize the risk of overfitting and getting incorrect results — we use the train part during *NN's* weights optimization and the test part to calculate final metrics. We use a three-layer perceptron with tanh activation functions and Adam [9] optimization algorithm using pytorch framework [10]. We evaluate our algorithm compared to a faiss IVF implementation [11] which is a current industrial standard using *SMAPE* and precision metrics — equations:

$$SMAPE(A,F) = 100 * \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i - F_i|}{|A_i + F_i|/2};$$

$$Precision = \frac{TP}{TP + FP}.$$

Where in our case A_i is the distance between *i*-th query vector q_i and its actual nearest neighbor from *X* and F_i is the distance between *i*-th query vector q_i and its suggested by algorithm approximate nearest neighbor from *X*. In other words, the SMAPE metric shows us how much the distances to the ground truth nearest neighbors and to the approximated neighbors differ on average.

In the case of the precision metric, we have TP — the number of cases where the approximate nearest neighbor equals the actual nearest neighbor and FP — the number of cases where the approximate nearest neighbor differs from the actual nearest neighbor. In other words, this metric shows us how often our approximated nearest neighbors exactly coincide with the ground truth ones.

We have final results presented in Tables 1, 2, 3. We also have a general structure of the result table:

– *X*-size — number of vectors in the indexed dataset;

– *Q*-size — number of vectors in the queries training set;

– *K* — number of buckets in the algorithm;

– *N*probe — number of adjoining buckets to use in the brute force phase in order to achieve a better precision;

– *IFV Prec.*/ *IFV SMAPE* — precision and *SMAPE* metrics of the faiss *IFV*;

– *SSIFV Prec.*/ *SSIFV SMAPE* — precision and *SMAPE* metrics of the novel semi-supervised *IFV* proposed in the paper.

Table 1

***X*-size**	***Q*-size**	***K***	***N*probe**	***IFV Prec.***	***IFV SMAPE***	***SSIFV Prec.***	***SSIFV SMAPE***
10*K*	10*K*	200	1	0.055	8.7%	0.083	7.7%
10*K*	10*K*	200	5	0.200	4.37%	0.255	3.69%
10*K*	10*K*	200	20	0.480	1.81%	0.524	1.56%
1*M*	10*K*	2000	1	0.063	7.41%	0.071	7.1%
1*M*	10*K*	2000	5	0.200	3.8%	0.220	3.72%
1*M*	10*K*	2000	20	0.435	1.79%	0.491	1.65%

$X \sim N(0,1)$; $Q \sim N(0,1)$ results

Table 2

***X*-size**	***Q*-size**	***K***	***N*probe**	***IFV Prec.***	***IFV SMAPE***	***SSIFV Prec.***	***SSIFV SMAPE***
10*K*	10*K*	200	1	0.057	8.68%	0.066	8.53%
10*K*	10*K*	200	5	0.197	4.40%	0.207	4.33%
10*K*	10*K*	200	20	0.473	1.87%	0.460	1.95%
1*M*	10*K*	2000	1	0.061	8.16%	0.069	7.99%
1*M*	10*K*	2000	5	0.218	4.32%	0.217	4.34%
1*M*	10*K*	2000	20	0.490	1.77%	0.498	1.77%

$X \sim Exponential\,(1)$; $Q \sim Exponential(1)$ results

Table 3

***X*-size**	***Q*-size**	***K***	***N*probe**	***IFV Prec.***	***IFV SMAPE***	***SSIFV Prec.***	***SSIFV SMAPE***
10*K*	10*K*	200	1	0.025	14.76%	0.137	3.87%
10*K*	10*K*	200	5	0.107	6.14%	0.403	1.46%
10*K*	10*K*	200	20	0.305	2.49%	0.756	0.41%
1*M*	10*K*	2000	1	0.035	11.68%	0.141	3.65%
1*M*	10*K*	2000	5	0.130	4.97%	0.419	1.28%
1*M*	10*K*	2000	20	0.341	2.44%	0.766	0.40%

$X \sim N(0,1)$; $Q \sim Exponential(1)$ results

CONCLUSION

The experimental results of our novel semi-supervised modification to the Inverted File (IVF) index approach for approximate nearest neighbor search look very promising, because SS-IVF approach outperforms the industry standard implementation in a lot of different experiment configurations from the raw precision/smape metrics perspective, especially in scenarios where query distribution significantly differs from the indexed dataset. However, this SS-IVF algorithm is still quite far from a production solution, since we have not yet done an efficient C/C++ implementation, which would use parallelization and low-level optimizations.

REFERENCES

1. P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, 1998.
2. H. Jégou, M. Douze, and C. Schmid, “Product Quantization for Nearest Neighbor Search,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/ document/5432202
3. G. Voronoi, “Une méthode géométrique pour la détermination des régions de visibilité dans le voisinage d’un point de l’espace (A geometric method for determining regions of visibility in the vicinity of a point in space),” *Journal de Mathématiques Pures et Appliquées (Journal of Pure and Applied Mathematics)*, 1908.
4. J. Johnson, M. Douze, and H. Jégou, “Optimizing Product Quantization for Nearest Neighbor Search,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/ document/6619223
5. D. E. Rumelhart and J. L. McClelland, “Learning Internal Representations by Error Propagation,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/ document/6302929
6. Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*. 2016. Available: https://www.deeplearningbook.org/
7. X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*. 2010. [Online]. Available: https://proceedings.mlr.press/v9/ glorot10a/glorot10a.pdf
8. D.P. Kingma, *Adam: A Method for Stochastic Optimization*. 2014. [Online]. Available: https://arxiv.org/abs/1412.6980
9. *PyTorch*. [Online]. Available: https://pytorch.org/
10. *faiss::IndexIVF Class Reference*. [Online]. Available: https://faiss.ai/cpp_api/struct/ structfaiss_1_1IndexIVF.html

Received 05.09.2023

INFORMATION ON THE ARTICLE

Anton A. Bazdyrev, ORCID: 0000-0001-8191-897X, Educational and Research Institute for Applied System Analysis of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine, e-mail: bazdyrev.anton@gmail.com

ПІДХІД З НАПІВКЕРОВАНИМ НАВЧАННЯМ В ІНВЕРТОВАНОМУ ФАЙЛОВОМУ ІНДЕКСІ ДЛЯ ПОШУКУ НАБЛИЖЕНОГО НАЙБЛИЖЧОГО СУСІДА / А.А. Баздирев

Анотація. Запропоновано удосконалення підходу з використанням інвертованого файлового індексу для пошуку наближених найближчих сусідів з використанням напівкерованого навчання та навчання з учителем з метою підвищення ефективності проміжної кластеризації та досягнення більш збалансованих розмірів кластерів. Запропонований метод полягає у створенні кластерів за допомогою нейронної мережі з розв’язанням завдання класифікації векторів запитів у той самий кластер, що і їхні відповідні найближчі сусідні вектори у вихідному наборі даних. У поєднанні з мінімізацією стандартного відхилення розмірів кластерів процес індексування стає більш ефективним і точним під час наближеного пошуку найближчих сусідів. Через емпіричну оцінку на тестовому наборі даних продемонстровано, що запропонований підхід до індексу виявився більш точним порівняно з індустрійно-стандартною реалізацією із фіксованими параметрами, включаючи загальну кількість кластерів та кількість кластерів, що виділяються для запитів. Метод перспективний для підвищення ефективності пошуку найближчих сусідів у великорозмірних наборах даних у різних застосуваннях, таких як інформаційний пошук, пошук за природною мовою, рекомендаційні системи тощо.

Ключові слова: пошук наближених найближчих сусідів, інвертований файловий індекс, дані високої розмірності, машинне навчання.