A Learning-to-Rank Formulation of Clustering-Based Approximate Nearest Neighbor Search

Thomas Vecchiato
Ca' Foscari University of Venice
Venice, Italy
880038@stud.unive.it

Claudio Lucchese
Ca' Foscari University of Venice
Venice, Italy
claudio.lucchese@unive.it

Franco Maria Nardini
ISTI-CNR
Pisa, Italy
francomaria.nardini@isti.cnr.it

Sebastian Bruch
Pinecone
New York, NY, USA
sbruch@acm.org

ABSTRACT

A critical piece of the modern information retrieval puzzle is approximate nearest neighbor search. Its objective is to return a set of k data points that are closest to a query point, with its accuracy measured by the proportion of exact nearest neighbors captured in the returned set. One popular approach to this question is clustering: The indexing algorithm partitions data points into non-overlapping subsets and represents each partition by a point such as its centroid. The query processing algorithm first identifies the nearest clusters—a process known as *routing*—then performs a nearest neighbor search over those clusters only. In this work, we make a simple observation: The routing function solves a ranking problem. Its quality can therefore be assessed with a ranking metric, making the function amenable to learning-to-rank. Interestingly, ground-truth is often freely available: Given a query distribution in a top-k configuration, the ground-truth is the set of clusters that contain the exact top-k vectors. We develop this insight and apply it to Maximum Inner Product Search (MIPS). As we demonstrate empirically on various datasets, learning a simple linear function consistently improves the accuracy of clustering-based MIPS.

CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**.

KEYWORDS

Approximate Nearest Neighbor Search; Inverted File; Learning to Rank

ACM Reference Format:
Thomas Vecchiato, Claudio Lucchese, Franco Maria Nardini, and Sebastian Bruch. 2024. A Learning-to-Rank Formulation of Clustering-Based Approximate Nearest Neighbor Search. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24), July 14–18, 2024, Washington, DC, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3626772.3657931

SIGIR '24, July 14–18, 2024, Washington, DC, USA

ACM ISBN 979-8-4007-0431-4/24/07
https://doi.org/10.1145/3626772.3657931

1 INTRODUCTION

Information Retrieval (IR) is no stranger to the problem of top-k retrieval. Inverted indexes, for example, have long been the backbone of exact and approximate search over *sparse* vectors [33], such as encodings of text using BM25 [28]. With the advent of representation learning and the proliferation of dense vectors as embeddings of text, a different form of approximate retrieval that originates in neighboring scientific disciplines has become a major part of modern IR: Approximate Nearest Neighbor (ANN) search [4].

The ANN setup should be familiar: A collection of vectors are processed into a data structure, known as the index, in an offline phase. When a query vector is presented, the ANN algorithm uses the index to quickly find the closest vectors to the query point, where closeness is defined based on the inner product between vectors or their Euclidean distance. Importantly, the returned set is often an approximate solution: If $\mathcal{S}$ is the exact set of k closest vectors to the query, and $\tilde{\mathcal{S}}$ is the set returned by the ANN algorithm, then the overlap between the two sets serves as a barometer quantifying the accuracy of the search: $|\mathcal{S} \cap \tilde{\mathcal{S}}|/k$.

This work concerns ANN over dense vectors, with a particular focus on clustering-based methods, where a collection is first split into L geometric partitions using a clustering algorithm. Each partition is then represented by a vector (e.g., its centroid). The list of partitions along with their centroids make up the index.

At search time, the algorithm first computes the similarity—in this work, inner product—between centroids and the query. It then identifies the ℓ closest partitions and performs ANN search over those partitions only. We call the first step "routing" where a query vector is mapped to ℓ partitions. Using $\ell \ll L$ reduces the search space dramatically, leading to an efficient search but at the expense of accuracy. In this work, we assume that search within clusters is exact; in practice that itself may be another ANN algorithm.

In that context, we observe a simple but consequential fact: Given a fixed partitioning of the data, the routing machinery is a ranking function. To see how, consider the top-1 case. Given a query, the algorithm ranks partitions by their likelihood of containing the nearest neighbor. As we assumed the clustering algorithm gives a *partitioning*—so that, a data vector belongs to one partition—a query will have a single "relevant" answer: the partition that contains its nearest neighbor. Therefore, routing quality can be determined with ranking metrics such as Mean Reciprocal Rank (MRR).

Perhaps more interesting is the fact that such a ranking function can be *learnt* using Learning-to-Rank (LTR) [5]. In fact, because MRR is an appropriate metric, and because each query has a single relevant answer, the cross-entropy loss [3, 7] is provably the right surrogate to optimize. Furthermore, training data is cheap: All one requires is a set of queries. The ground-truth "relevance" labels can be computed by performing an exact search for each query, finding the nearest neighbor, and identifying the partition it belongs to.

It appears that we have all the ingredients to learn a routing function for a fixed partitioning of vectors. In this work, we develop that insight into a working solution. We show through experiments on embeddings of various text datasets, that learning a routing function can boost the accuracy of ANN search. In fact, the gains are consistent regardless of our choice of the clustering algorithm.

What is even more appealing about our results is that we simply learn a linear function: $\tau(q;\ W) = Wq$, where $q \in \mathbb{R}^d$ and $W \in \mathbb{R}^{L\times d}$. The output of $\tau(\cdot)$ gives us the ranking scores for each of the L partitions. Pleasantly, that means that the i-th row of W can be thought of as a learnt representative vector of the i-th partition. That entails, we can conveniently plug in the learnt representatives into the original routing machinery without any change whatsoever to implementation—an important criterion for production systems.

Our findings show the potential LTR holds for ANN search. We are excited to bridge these two fields in our work and thereby motivate the community to explore this junction. To that end, we briefly review the literature in Section 2, describe our method in Section 3, and present experimental results in Section 4. We conclude this work in Section 5 with remarks on possible future directions.

2 RELATED WORK

ANN algorithms come in many flavors including trees [2, 13], hash tables [16], and graphs [19, 23]. The method relevant to this work is the clustering-based approach, also known as Inverted File (IVF) [20]. As we intend to keep this review concise, we refer the interested reader to [4] for a thorough treatment of the subject.

As explained in Section 1, indexing in clustering-based ANN utilizes a clustering algorithm to partition the vectors. A typical choice [1, 20] is standard KMeans, which iterates between an expectation phase (computing partition centroids given current cluster assignments) and an assignment step (computing cluster assignments given centroids). This choice makes sense especially in ANN with Euclidean distance, as it can be shown that the KMeans objective helps find Voronoi regions—a crucial structure in ANN search.

In this work, we also use two alternative clustering strategies. One is a variant of KMeans known as Spherical KMeans [14], where after each expectation phase, the centroids are L_2-normalized and projected onto the unit sphere. This small modification makes the clustering algorithm arguably more suitable for ANN with angular distance and has been shown [6] to perform well for inner product.

The final clustering method used in this work is [12], which we refer to as Shallow KMeans, and can be thought of as the first iteration of KMeans. The algorithm begins by selecting L data points uniformly at random and using them as cluster representatives. It then assigns data points to partitions by inner product with the cluster representatives: the partition whose representative yields the largest inner product with a data point becomes its home partition.

We now briefly review LTR, a well-researched topic in IR. In general, given a set of training queries with ground-truth (relevance) labels, and a collection of items, the goal is to learn a function that computes a score for any query-item pair. These scores induce an order among the items, resulting in a ranking. The rankings generated by a learnt function ideally maximize a ranking metric for a query distribution of interest, such as NDCG [18] or MRR.

Because ranking metrics are discontinuous or otherwise flat almost everywhere, supervised LTR resorts to optimizing surrogate objectives [8–11, 21, 26, 29–31, 34, 35]. One objective that is appropriate for our work is the cross entropy-based loss described in [3]. It has been shown [3, 7] that the cross-entropy loss is a consistent surrogate for MRR when each query has at most one relevant item with probability 1. Noting that our setup satisfies these conditions, as explained in Section 1, we will use this ranking loss in this work.

3 METHODOLOGY

Let $\langle u, v\rangle$ denote the inner product of two vectors $u, v \in \mathbb{R}^d$. Given a collection of datapoints $\mathcal{X} \subset \mathbb{R}^d$ and a query point $q \in \mathbb{R}^d$, the (top-1) Maximum Inner Product Search (MIPS) problem is to find:

$$u^* = \arg\max_{u\in\mathcal{X}} \langle q, u\rangle\,. \tag{1}$$

As noted earlier, to solve this problem efficiently (albeit approximately), one prominent approach clusters $\mathcal{X}$ into a set of non-overlapping partitions $\{C_i\}_{i=1}^L$, where $C_i \in 2^{\mathcal{X}}$, each represented with a vector μ_i. A routing function $\tau : \mathbb{R}^d \to \mathbb{R}^L$ then computes a "goodness" score for each C_i. Typically, $\tau = Mq$ where $M \in \mathbb{R}^{L\times d}$ whose i-th row is μ_i. Subsequently, only the points within the top ℓ clusters by score are evaluated to find the solution to Equation (1).

We claim in this work that the function $\tau(\cdot)$ can be learnt using supervised LTR methods. Concretely, we wish to learn $\tau(q;\ W) = Wq$ where $W \in \mathbb{R}^{L\times d}$. We choose to learn a linear function so that it can trivially replace $\tau = Mq$ by setting $M = W$ without any change to the framework. However, the method can be extended to any other parameterized family of functions.

To complete the picture, we must define a training dataset and a loss function. We build training examples as follows. For query q, let τ^* be the oracle routing function such that the i-th component of $\tau^*(q)$ is 1 if $u^* \in C_i$ and 0 otherwise. The training set then comprises pairs $(q_i, \tau^*(q_i))$, where $q_i \in Q$ is a query point.

We learn $\tau(\cdot)$ by maximizing MRR. Note that MRR is especially meaningful, as opposed to classification metrics, in the more general case where we allow probing $\ell > 1$ partitions. In this case, promoting a ranking function that places the relevant cluster among the top positions increases the probability of finding u^*.

We use the cross-entropy loss to maximize MRR [3, 7]. The loss for a single query q reduces to the following:

$$-\sum_{i=1}^{L} \tau_i^* \log \frac{\exp(\tau_i)}{\sum_{j=1}^{L}\exp(\tau_j)},$$

where, with a slight abuse of notation, we write τ_k and τ_k^* to denote the k-th component of $\tau(q)$ and $\tau^*(q)$, respectively. Note that, because τ^* has a single non-zero component, the sum collapses to a single term. The mean of the loss above over the entire training query set becomes the final objective for the optimization problem.

3.1 Generalizing to top-k

What we have described thus far concerns top-1 MIPS only. That is, for each query, we have a single correct partition to route it to, and we wish to learn said function. We use this setup in the rest of this work, but note that learning a routing function that optimizes for top-k with $k > 1$ is straightforward. Let us explain why.

Suppose $k > 1$ and let $\mathcal{S}$ denote the (exact) set of top-k data vectors for query q. Then one example of an oracle routing function $\tau^*(q)$ will have 1 in its i-th component if and only if $\mathcal{S} \cap C_i \neq \emptyset$, and 0 otherwise. The loss function, for query q, can then be updated to follow its general from from [3] as follows:

$$-\sum_{i=1}^{L} \frac{2^{\tau_i^*} - \gamma_i}{\sum_{j=1}^{L} 2^{\tau_j^*} - \gamma_j} \log \frac{\exp(\tau_i)}{\sum_{j=1}^{L} \exp(\tau_j)},$$

where, as before, τ_k^* and τ_k denote the k-th components of $\tau^*(q)$ and $\tau(q)$, and γ_k's are uniformly sampled from the unit interval.

4 EXPERIMENTS

We now report our experimental evaluation by first detailing the methodology and then presenting and discussing the results.

4.1 Experimental Setup

Datasets. We use three publicly-available datasets: Ms Marco [25], HotpotQA [36], and Fever [32]. Ms Marco Passage Retrieval consists of about 8.8 million short passages and 909,000 train queries. HotpotQA is a question answering dataset collected from the English Wikipedia, consisting of 5.2 million documents and 98,000 queries. Finally, Fever has 5.4 million documents and 13,000 queries.

Embedding Models. We use embedding models to transform queries and documents into d-dimensional dense vectors. The models include `tasB`[1] [15] (d=768), and `contriever`[2] [17] (d=768). We also include models fine-tuned by [27] on 1 billion text pairs: `all-MiniLM-L6-v2`[3] (d=384), `all-mpnet-base-v2`[4] (d=768), and `all-distilroberta-v1`[5] (d=768).

Evaluation Metric. We evaluate all methods in terms of top-k *accuracy*, defined as follows. For each query, we obtain the top ℓ partitions according to $\tau(\cdot)$, and record the percentage of the top-k documents contained in those partitions.

Baseline. We evaluate the accuracy of a learnt $\tau(\cdot)$ against the baseline routing function $\tau(q) = Mq$ where the i-th row of M is simply the representative of the i-th partition, μ_i, returned by the clustering algorithm. Specifically, we identify the top ℓ partitions for q according to Mq, and measure its accuracy as defined above.

Implementation Details. We model $\tau(\cdot;\ W)$ as a linear function with $W \in \mathbb{R}^{L \times d}$. To compute the loss function, we transform Wq with softmax, which gives the probability of a query being routed to each partition. We use Adam [22] with a learning rate of 10^{-4} to optimize the loss function. We set the batch size to 512 and train for a maximum of 100 epochs.[6]

[1] https://huggingface.co/sentence-transformers/msmarco-distilbert-base-tas-b
[2] https://huggingface.co/facebook/contriever
[3] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[4] https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[5] https://huggingface.co/sentence-transformers/all-distilroberta-v1
[6] Our code is available at: https://github.com/tomvek/mips-learnt-ivf

Table 1: Top-1 accuracy of baseline and learnt routing functions on `all-MiniLM-L6-v2` embeddings. Columns are blocked by the clustering algorithm. For each dataset-clustering pair, we measure top-1 accuracy by setting ℓ to 0.1% and 1% of the number of partitions, L.

Dataset	Method	Standard		Spherical		Shallow	
		0.1%	1%	0.1%	1%	0.1%	1%
Ms Marco	Baseline	0.392	0.779	0.627	0.869	0.517	0.815
	Learnt	0.746	0.940	0.751	0.938	0.670	0.923
HotpotQA	Baseline	0.089	0.481	0.328	0.684	0.258	0.724
	Learnt	0.488	0.844	0.493	0.833	0.412	0.827
Fever	Baseline	0.102	0.443	0.249	0.562	0.279	0.621
	Learnt	0.663	0.865	0.662	0.872	0.633	0.912

Given a collection of m vectors (i.e., embeddings of a dataset), we take the following steps in each experiment. We first run one of the three clustering algorithms (standard, spherical, and shallow KMeans) on the data vectors, clustering them into $L = \sqrt{m}$ partitions. We then split the query set into training (60%), validation (20%), and test (20%) queries, and construct training, validation, and test examples using the procedure described in Section 3. Finally, we train on the training examples and take the best model according to the loss on the validation examples. We evaluate baseline and learnt functions on the test splits.

4.2 Experimental Results

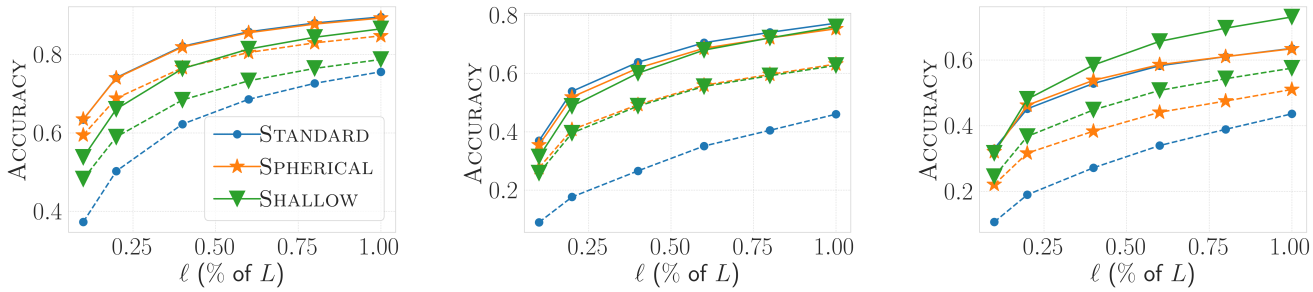
Top-1. Table 1 presents the top-1 accuracy for baseline and learnt routing functions on the `all-MiniLM-L6-v2` embeddings of text datasets. As the table shows, we consider three configurations where the partitions are produced by standard, spherical and shallow KMeans. For each configuration, we measure accuracy by setting ℓ to 0.1% and 1% of L, the total number of partitions.

As is evident from Table 1, a learnt $\tau(\cdot)$ consistently outperforms the baseline. Taking standard KMeans and $\ell = L/100$ as an example, learning improves accuracy by about 21% on Ms Marco, 75% on HotpotQA, and 95% on Fever. The difference is greater for smaller values of ℓ, indicating that the closest partitions to query vectors are of much greater quality. That makes sense as our learning objective heavily focuses on the top partition.

We make another observation that may be of independent interest: The shallow KMeans clustering algorithm performs remarkably well for MIPS. Considering the fact that shallow KMeans requires no training whatsoever—clustering is a matter of sampling L points from the dataset followed by assignment—coupled with its high accuracy, it makes for an efficient yet effective clustering in practice.

Top-k. Even though, when learning a routing function, we optimize a loss that is only concerned with top-1 accuracy, we investigate in this section the impact of such training on top-k accuracy. Does learning to optimize top-1 accuracy benefit top-k retrieval for $k > 1$? The answer appears to be in the affirmative, as the results of our experiments show in Figure 1, rendering top-10 accuracy as a function of ℓ. Trends are similar for $k = 100$.

These results are encouraging and bode well for an extension of the loss function to top-k retrieval, as explained in Section 3.1. We leave an exploration of that generalization to future work.



(a) Ms Marco (b) HotpotQA (c) Fever

Figure 1: Top-10 accuracy as a function of ℓ (expressed as percent of total number of partitions, L), on the `all-MiniLM-L6-v2` embeddings. In all figures, the dashed lines indicate the baseline and the solid lines show the performance of the learnt routing function.

Table 2: Top-1 accuracy of baseline and learnt routing on Ms Marco. For each embedding model, the top row represents the baseline accuracy and the bottom row the learnt function's. As before, we report top-1 accuracy by setting ℓ to 0.1 and 1 percent of L.

Encoding	Standard		Spherical		Shallow	
	0.1%	1%	0.1%	1%	0.1%	1%
`tasB`	0.480	0.835	0.680	0.915	0.553	0.869
	0.727	0.936	0.724	0.933	0.612	0.896
`contriever`	0.602	0.895	0.756	0.938	0.640	0.909
	0.790	0.952	0.780	0.946	0.690	0.927
`all-mpnet-base-v2`	0.763	0.952	0.794	0.958	0.691	0.940
	0.818	0.967	0.819	0.966	0.733	0.951
`all-distilroberta-v1`	0.752	0.955	0.779	0.960	0.664	0.935
	0.807	0.967	0.806	0.966	0.706	0.945

Other Embeddings. We repeat this exercise for other embedding models. Table 2 reports the top-1 accuracy for select values of ℓ on Ms Marco. We observe that the general trend from before holds: Learning does lead to gains in accuracy, especially when ℓ is small. The McNemar's test [24] suggests statistically significant difference between the methods (p-value < 0.001).

Unlike the previous results, gaps between the baseline and learnt routing are smaller. We believe what contributes to smaller gains has to do with the larger dimensionality of the embedding vectors (768 of models in Table 2 versus 384 of `all-MiniLM-L6-v2`) and the capacity of the linear function in learning appropriate partition representatives. This is a question we leave to future work.

5 CONCLUSIONS AND FUTURE WORK

Formalizing routing in a clustering-based ANN algorithm as ranking is intuitive enough. When a partitioning has been finalized, the task is to rank partitions by their likelihood of containing the nearest neighbor to a query. Building on that intuition, we explored the feasibility of employing LTR algorithms to improve routing. As we argued in this work, preparing the ingredients necessary to apply LTR to routing, such as labeled training data, requires a modest effort—in fact, all that is needed is a set of queries.

We empirically assessed the advantages of learning a (linear) routing function and demonstrated the potential LTR holds for clustering-based ANN. We observed that, generally, learning leads to gains in accuracy. In other words, for the same number of partitions probed, a learnt function routes the query to better partitions a higher percentage of the time.

We have only scratched the surface in this exploratory work, with many interesting questions left to investigate in follow-up studies. As hinted earlier, we wish to understand the impact of the more general loss function of Section 3.1 on top-k retrieval. Additionally, we hope to shed more light on the smaller gains on high-dimensional embeddings. Finally, we intend to extend the method to other metrics such as Euclidean or angular distance.

More exciting still is the question of the utility of LTR in the partitioning algorithm itself. Recall that, throughout this work, we assumed that a collection of vectors has already been partitioned by some clustering algorithm. We then "refined" the representative vectors through supervised learning. What is clear, however, is that such learning can be incorporated into the clustering phase itself. In other words, during clustering, one may iteratively refine the cluster representatives and update cluster assignments accordingly. This opens the door to query-aware clustering for ANN search—an area that has remained under-explored.

Acknowledgements. This work was partially supported by the Horizon Europe RIA "Extreme Food Risk Analytics" (EFRA), grant agreement n. 101093026, by the iNEST - Interconnected Nord-Est Innovation Ecosystem (iNEST ECS_00000043 – CUP H43C22000540006) and the PNRR - M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI" both funded by the European Commission under the NextGeneration EU program. The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the European Union, nor can the European Union be held responsible for them.

REFERENCES

[1] Alex Auvolat, Sarath Chandar, Pascal Vincent, Hugo Larochelle, and Yoshua Bengio. 2015. Clustering is Efficient for Approximate Maximum Inner Product Search. arXiv:1507.05910 [cs.LG]

[2] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (9 1975), 509–517.

[3] Sebastian Bruch. 2021. An Alternative Cross Entropy Loss for Learning-to-Rank. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia). 118–126.

[4] Sebastian Bruch. 2024. *Foundations of Vector Retrieval.* Springer Nature Switzerland.

[5] Sebastian Bruch, Claudio Lucchese, and Franco Maria Nardini. 2023. Efficient and Effective Tree-based and Neural Learning to Rank. *Foundations and Trends in Information Retrieval* 17, 1 (2023), 1–123.

[6] Sebastian Bruch, Franco Maria Nardini, Amir Ingber, and Edo Liberty. 2023. Bridging Dense and Sparse Maximum Inner Product Search. arXiv:2309.09013 [cs.IR]

[7] Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2019. An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval* (Santa Clara, CA, USA). 75–78.

[8] Sebastian Bruch, Masrour Zoghi, Mike Bendersky, and Marc Najork. 2019. Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval.*

[9] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*. 89–96.

[10] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report MSR-TR-2010-82. Microsoft Research.

[11] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*. 129–136.

[12] Flavio Chierichetti, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi, and Eli Upfal. 2007. Finding near Neighbors through Cluster Pruning. In *Proceedings of the 26th ACM SIGMOD Symposium on Principles of Database Systems* (Beijing, China). 103–112.

[13] Sanjoy Dasgupta and Kaushik Sinha. 2015. Randomized Partition Trees for Nearest Neighbor Search. *Algorithmica* 72, 1 (5 2015), 237–263.

[14] Inderjit S. Dhillon and Dharmendra S. Modha. 1999. *Concept Decompositions for Large Sparse Text Data using Clustering.* Technical Report RJ 10147. Array.

[15] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada). 113–122.

[16] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA). 604–613.

[17] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research* (2022).

[18] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.

[19] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems*, Vol. 32.

[20] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[21] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 217–226.

[22] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[23] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (4 2020), 824–836.

[24] Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 12 (6 1947), 153–157.

[25] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. (November 2016).

[26] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval* 13, 4 (2010), 375–397.

[27] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3982–3992.

[28] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3.. In *TREC (NIST Special Publication, Vol. 500-225)*, Donna K. Harman (Ed.). National Institute of Standards and Technology (NIST), 109–126.

[29] Cynthia Rudin. 2009. The P-Norm Push: A Simple Convex Ranking Algorithm That Concentrates at the Top of the List. *Journal of Machine Learning Research* 10 (Dec. 2009), 2233–2271.

[30] Cynthia Rudin and Yining Wang. 2018. Direct Learning to Rank and Rerank. In *Proceedings of Artificial Intelligence and Statistics AISTATS.*

[31] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. SoftRank: Optimizing Non-smooth Rank Metrics. In *Proceedings of the 1st International Conference on Web Search and Data Mining*. 77–86.

[32] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a Large-scale Dataset for Fact Extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 809–819.

[33] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2018. Efficient Query Processing for Scalable Web Search. *Foundations and Trends in Information Retrieval* 12, 4–5 (Dec 2018), 319–500.

[34] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*. 1192–1199.

[35] Jun Xu and Hang Li. 2007. AdaRank: A Boosting Algorithm for Information Retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* 391–398.

[36] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2369–2380.