Data-Driven Learned Metric Index: An Unsupervised Approach

Terézia Slanináková(✉), Matej Antol, Jaroslav Oľha, Vojtěch Kaňa, and Vlastislav Dohnal

Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
{xslanin,matejantol,olha,456598,dohnal}@mail.muni.cz

Abstract. Metric indexes are traditionally used for organizing unstructured or complex data to speed up similarity queries. The most widely-used indexes cluster data or divide space using hyper-planes. While searching, the mutual distances between objects and the metric properties allow for the pruning of branches with irrelevant data – this is usually implemented by utilizing selected anchor objects called pivots. Recently, we have introduced an alternative to this approach called Learned Metric Index. In this method, a series of machine learning models substitute decisions performed on pivots – the query evaluation is then determined by the predictions of these models. This technique relies upon a traditional metric index as a template for its own structure – this dependence on a pre-existing index and the related overhead is the main drawback of the approach.

In this paper, we propose a data-driven variant of the Learned Metric Index, which organizes the data using their descriptors directly, thus eliminating the need for a template. The proposed learned index shows significant gains in performance over its earlier version, as well as the established indexing structure M-index.

Keywords: Index structures · Learned index · Unstructured data · Content-based search · Metric space · Machine learning

1 Introduction

Searching within collections of unstructured or complex data (such as images, audio files or protein structures) is a challenging task. Whereas in structured data-sets, the order of the data objects is determined using a straightforward key (e.g., their alphabetical order) and the match to a search filter is objectively given (e.g., retrieve all records where `created_on` $\leq$ `2010-04-01`), in the realm of unstructured data, such properties do not exist. Since there is no intrinsic

This research has been supported by the Czech Science Foundation project No. GA19-02033S. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

ordering to the data, there is no single agreed-upon response to a given search query.

The issue can be addressed using metric spaces, where the pairwise similarity of objects can be leveraged to organize the data and formulate search queries. If we can design a suitable distance function that meets certain criteria (such as symmetry and triangle inequality), any indexing structure or search algorithm designed for generic metric spaces can be applied to our data, and various pruning rules can be used to reduce the search space.

The search itself is usually performed using various similarity queries, wherein we specify a query object and choose the properties of the desired result in relation to this object (e.g., the k closest objects to the query object – kNN query, or all objects within a certain range from the query object – range query). Even after applying the metric spaces and similarity searching methods, a major challenge remains – since these complex data-sets tend to have a very high number of intrinsic dimensions [5], the distance computations needed for index construction and query evaluation are computationally expensive.

This problem can be addressed using an alternative approach – finding the similarity in large groups of data can be reformulated into a pattern searching task, which can be solved by machine learning. We have previously introduced such a solution, using supervised machine learning to imitate the structure of a pre-existing index, resulting in a hierarchy of several learned models that we call *Learned Metric Index (LMI)* [2]. While this approach achieves very good performance in the query evaluation phase by eliminating costly distance computations, its main downside is obvious – to train such an index, we first need to construct one of the traditional index structures as a template.

In our current work, we have evolved the LMI's approach beyond the need for a pre-existing index built using traditional methods. Instead, we can now construct the LMI from scratch, using nothing but the pattern recognition capability of the machine learning models to discern the natural distributions of the data in the metric space.

To the best of our knowledge, this is a completely novel method for tackling the problem of indexing unstructured data. This paper describes our approach and implementation in detail and evaluates its performance, comparing it to the traditional state-of-the-art indexes and our previous implementations of supervised learned indexing.

2 Related Work

More and more research work has recently addressed the possibilities of enhancing or even replacing standard database index models (B^+-trees) with machine learning [8,14,17]. The authors argue that machine learning models can be trained for the same purpose of answering queries (categorizing a query object to the most suitable class, which represents a child node) while presenting several performance benefits.

For instance, in inverted indexes, a hierarchical machine learning model is used to reduce index size at the expense of performance [29,34]. In cases involving multidimensional data, learned indexes attempt to approximate the search to be reasonably efficient. The density distribution of multidimensional data is approximated to create a new index structure in [31,33]. Another application of learned models [24] presents an index named Flood that creates not only a consistently performing index for multidimensional data, but also optimizes both index and data storage layout. In [18], a learned variation of Bloom filters for multidimensional data can save a significant portion of space. A wide study [16] of various algorithms for kNN queries over multidimensional Euclidean spaces concludes that it is still a research challenge to provide a solution of highly precise approximate kNN search due to the curse of dimensionality.

We carry on with the proposition of utilizing machine learning to index structured data and apply it to complex data and metric space model. In this paper specifically, we follow up on the Learned Metric Index method we introduced in [2]. Even though we believe that our research is original, the idea of learned models has been applied before in the domain of similarity searching in metric data. In [13], ANN-tree was introduced to solve the 1-NN problem for metric space scenarios. Authors of [22] consequently introduced the FLANN library to perform the 1-NN search significantly faster than a previous, nearly brute force implementation.

Recently, a new partitioning procedure focused on nearest neighbor search performance, called Neural Locality-Sensitive Hashing (Neural LSH) [7], has been shown to outperform traditional partitioning methods (k-means) consistently. A learned model that approximates bounds on k nearest neighbor distances and consequently allows precise and memory-efficient computation of reverse nearest neighbors has been introduced in [4]. The authors conducted experiments on up to 8-dimensional and low-volume data. Finally, Hünemörder et al. [11] explored the application of various predictive models to learn an index for approximate nearest-neighbor queries. Their evaluation on synthetic data as well as the MNIST data-set further demonstrates the research potential of this topic.

3 Indexing in Metric Spaces

A *metric space* $\mathcal{M} = (\mathcal{D}, d)$ is defined over a universe $\mathcal{D}$ of data objects and a distance function $d(\cdot, \cdot)$ that satisfies metric postulates. A database $X \subseteq \mathcal{D}$ of data objects forms a collection to be queried by a *k-nearest neighbors query* ($kNN(q)$ – k objects closest to the query object q), or the *range query* ($range(q, r)$ – all database objects closer to q than the distance r).

To avoid tedious sequential scanning, which is costly on large data-sets or with an expensive distance function, various indexing structures have been developed. Firstly, hierarchical structures include variations of the original M-tree [6], Spatial Approximation Trees [25], or Rank Cover Trees [10]. These structures divide data objects into groups or clusters, respecting their distribution in space.

They provide sub-linear search time $\mathcal{O}(n^{\alpha})$, where $\alpha \leq 1$ depends on data distribution. Next, permutations of preselected anchor objects (pivots) and their prefixes define (Voronoi-like) space cells at bounded costs, so M-index [26] and PPP-Codes [28] improve search efficiency substantially. Rearrangement of such cells is applied in [1,21]. Lastly, independent filtering techniques can be applied to further eliminate accessing excessive amounts of data objects, e.g. Binary Sketches for Secondary Filtering [19].

The properties of the metric function (namely symmetry and triangle inequality) are typically indispensable for constructing index structures and for the correctness of search. Learned indexes do not inherently depend on these properties, so the query evaluation based on predictions can be advantageous for non-metric distance functions as well.

4 Learned Metric Index

Learned Metric Index (LMI), as introduced in [2], is a hierarchical tree index structure of nodes containing machine learning models. These models are trained to search for (i.e., categorize) query objects, which emulates the behaviour of traditional index nodes. However, instead of determining the objects' positions according to their distances, a query is resolved by applying a series of predictions. This changes the standard paradigm of index building and query evaluation, resulting in very different performance characteristics and outperforming traditional similarity searching methods in many cases, both in terms of efficiency and effectiveness.

In general, the concept of LMI can be realized in two distinct ways. The first one involves using a pre-existing index and its data partitioning as labels for *supervised* training. In such a case, each data object has a label corresponding to its position in the original index, i.e., a concatenated list of integer values per index level. We have examined this variant in [2] and demonstrated that it can achieve more than competitive performance with state-of-the-art methods.

The other option is to assemble LMI "from scratch" by letting it create its own meaningful divisions of the data. Such approach exploits the information embedded in the descriptors of data objects to emulate the similarity function. This constitutes an *unsupervised* learning problem, which is the subject of this paper.

4.1 Training Unsupervised LMI

Training an unsupervised LMI requires: (i) digital fingerprint of objects to train on, and (ii) the number of clusters each model is expected to create, which defines the shape of the learned index structure. The training procedure of the whole LMI then starts with the root model, which is trained on the entirety of the given data-set, while its descendants are trained on smaller and smaller portions of the data as we dive deeper into the structure. The training is therefore sequential – the input of every model depends on the output of its parent.

Algorithm 1: Unsupervised Learned Metric Index training

```
Input: a data-set $X$, max. depth $H$ (tree height),
       max. number of children per level $A[]$
Output: a tree of trained models $T[][]$
part[1][1] = $X$;
for $lvl \leftarrow 1$ to $H$ do
    for $chld \leftarrow 1$ to $A[lvl]$ do
        if $part[lvl][chld] = \emptyset$ then
            continue
        end
        $M \leftarrow$ new model trained on part[$lvl$][$chld$] clustering the data into $A[lvl]$
        groups;
        if $lvl < H$ then
            for $obj \in part[lvl][chld]$ do
                p = $M$.predict($obj$);
                part[$lvl$+1][p].add($obj$);
            end
        end
        $T[lvl][chld] = M$;
    end
end
return T;
```

Algorithm 1 formally describes the entire training procedure. During the training, each model is presented with a clustering problem. The objective is to organize the data into a pre-specified number of groups according to their mutual similarity obtained from the descriptors. Each training epoch re-organizes the data to allow mutually similar objects to end up in the same cluster. A single instance of LMI is then created by connecting the parent models with their children, resulting in a tree structure.

4.2 Searching in LMI

We define the overall goal of LMI as finding as many of the query's k nearest neighbors as possible in the shortest time. The output of every learned model in the searching (inference) phase is a probability distribution, which can be viewed as the query's correspondence to each of the classes (i.e., child nodes). We expect LMI to be able to assign higher probabilities (and therefore higher search priorities) to categories where the query object and its nearest neighbors reside. The priority queue can then be formed in a naïve manner by sorting the child nodes based on the probabilities assigned by their parent model. This contrasts with traditional indexing methods, which need to calculate the distances to all of the child objects to form their priority queues.

The searching process of LMI is shown in Fig. 1. From the LMI's point of view, an answer to a query is gradually updated with objects from the visited leaf nodes. Note that the small sub-sections of the data-set contained within the

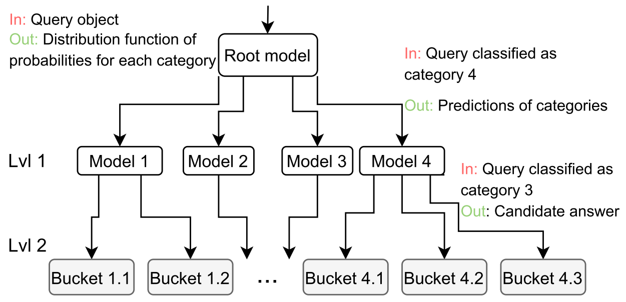
In: Query object
Out: Distribution function of probabilities for each category
Root model
In: Query classified as category 4
Out: Predictions of categories
Lvl 1
Model 1
Model 2
Model 3
Model 4
In: Query classified as category 3
Out: Candidate answer
Lvl 2
Bucket 1.1
Bucket 1.2
...
Bucket 4.1
Bucket 4.2
Bucket 4.3

Fig. 1. Example of a few initial steps of searching within a two-level LMI with four models on Level 1. The search continues until a stop condition is met.

leaf nodes are searched linearly – once a leaf node gets to the top of the priority queue, all of its objects are evaluated (i.e., added to the answer or discarded based on their distance) and the leaf node is removed from the queue.

4.3 Machine Learning Models

In the previous sections, we introduced a basic version of unsupervised LMI wherein we can use the machine learning models to build the index, and then use the probability outputs of these models to search the resulting structure.

However, in practice, very few unsupervised algorithms can operate probabilistically. To use a non-probabilistic unsupervised algorithm, we need to modify the approach in one of two ways. The first option is to build the structure using Algorithm 1, and use distance calculations for searching in the case of distance-based algorithms. The second option is to substitute the distance function with a supervised machine learning model. However, this second approach requires a modification of the building phase described by Algorithm 1, splitting the training into two steps[1].

We selected two basic machine learning algorithms to implement unsupervised LMI – K-Means and Gaussian Mixture Models (GMM).

K-Means is a well-established distance-based algorithm, which requires the Euclidean space to suitably place cluster centers – so-called *centroids* – within the data. The algorithm runs until a local optimum is reached by iteratively recalculating the centroids' position to minimize the sum of squares within clusters. Logistic Regression was selected as the supervised algorithm for the two-step version of this process.

Gaussian Mixture Model (GMM) employs a more flexible approach to data modelling, using soft clustering instead of the hard cluster assignments

[1] This training procedure consists of two separate phases: one for clustering the data, and the second for their categorization. For every level, the data is firstly clustered in the same way as described above. Subsequently, a supervised categorization machine learning algorithm is trained on the relevant portion of the data and the clustered labels.

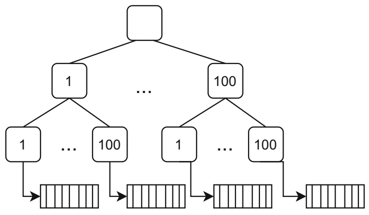

Fig. 2. Index architecture: 2 levels and 100 categories per model for the 1-million data-sets, and reduced to 71 categories for MoCap, since it is smaller.

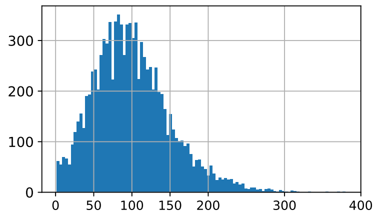

Fig. 3. Histogram of object distribution in the buckets (distance-based K-Means, Profiset).

made by K-Means. As its name suggests, GMM assumes that each data point could have been generated by any number of its k Multi-variate Gaussian distributions (k being the chosen number of clusters) with a given probability. To evaluate this probability, we must approximate the posterior probability of an object belonging to a cluster, given the observed data. *Bayesian GMM* is an extension of GMM, which estimates the object's cluster membership by Bayesian Variational Inference instead of calculating the marginal probabilities.

As a result, we evaluate four separate algorithms in the experimental phase – distance-based K-Means, K-Means with Logistic Regression, GMM and Bayesian GMM. The index-building and searching operations were implemented in Python, and algorithms used to prototype unsupervised LMI came from the scikit-learn library [30] with the exception of K-means with Logistic regression, where we employed an efficient GPU implementation of K-means [12].

5 Experiments

We have executed a wide range of experiments with three different multimedia data-sets: *CoPhIR*, *Profiset* and *MoCap*. CoPhIR [3] is a data collection of 282-dimensional vectors derived from five visual descriptors of images. Profiset [27] is a series of 4096-dimensional vectors extracted from Photo-stock images using a convolutional network. Finally, MoCap is HDM05 data-set [23] that consists of sequences of 3D skeleton poses, which were segmented to extract 4096-dimensional descriptors using AlexNet [15]. The data-set sizes were fixed at 1-million objects for CoPhIR and Profiset. MoCap contains 354,893 segments.

In contrast with the supervised version of LMI, unsupervised LMI has a unique architectural flexibility provided by the unsupervised mode of training, where one can specify the index architecture via the number of clusters per each model and thus optimize the performance. As a results, we chose to use a single architectural configuration throughout the experiments, consisting of two levels with a fixed number of nodes, as detailed in Fig. 2. As opposed to the traditional indexing structures, such as M-tree or M-index, LMI does not limit leaf node capacity. However, this fact does not cause the distribution of objects within buckets to be uncontrollably skewed, as Fig. 3 shows. The vast majority of the

Table 1. Building costs of various unsupervised setups and baselines. Unsupervised experiments were executed on a machine with 1 CPU – Intel Xeon E5-2650v2 2.60 GHz. K-Means (LR) utilized GPU - nVidia Tesla T4 16 GB. LMI baseline used Intel Xeon Gold 6230 2.10 GHz. M-index baseline used Intel Xeon E5-2620 2.00 GHz.

		Bayesian GMM	GMM	K-means (LR)	K-means (dist.)	Baselines	
						LMI	M-index
Build t. (h)	CoPhIR	0.305	0.351	1.926	0.639	2.670	0.330
	Profiset	1.419	1.553	9.554	3.698	0.230	0.490
	MoCap	0.351	0.467	2.200	0.627	0.390	0.170
Memory (gb)	CoPhIR	10.0	13.6	15.6	8.6	150.0	3.4
	Profiset	74.4	86.6	75.0	85.0	150.0	20.7
	MoCap	55.5	71.0	32.0	49.0	85.0	6.4

bucket occupancies is within the 75–125 interval, guaranteeing similar sequential search costs in the final part. This property allows us to skip searching of the leaf nodes in evaluation, and focus on the performance of the internal index navigation, where the various indexes truly differ.

In each of the experiments, we perform a 30-NN query for 1,000 randomly chosen query objects. The performance is measured in terms of recall – i.e., how many of the actual 30 nearest neighbors are returned when visiting a limited portion of data-set. We set such search limits (*stop-conditions*) as increasing thresholds spanning from 0.05% of the indexing structure searched (the lowest stop-condition) to 75% searched (the highest stop-condition).[2] As is the case with all indexes, we are primarily interested in optimizing the trade-off between *recall* and the searching time (i.e., the time needed to evaluate a query).

5.1 Building Costs

To provide a clear comparison of various indexes, we have to consider the costs of their construction. Table 1 documents the RAM usage and time required to build each of LMIs and M-index.

The table shows that the construction cost of a given setup is strongly influenced by the data-set dimensionality, which is consistent with the results observed in [2]. Specifically, the dimensionality of Profiset and MoCap descriptors is almost 15 times that of the CoPhIR data-set (282 vs 4096 features), which results in greater memory and building time requirements. The amount of the data present in the data-set influences the cost as well – in the case of MoCap, the number of objects the structure has to index is about one-third of the amount of Profiset, resulting in shorter building times and lower memory requirements.

[2] Full enumeration of stop-conditions used: 0.05%, 0.1%, 0.3%, 0.5%, 1%, 5%, 10%, 20%, 30%, 50% and 75% of the data-set size.

Table 2. The hyperparameters and their various settings for all four implemented algorithms. The highlighted values enabled the models to reach the best performance on majority of the data-sets. For further details, see documentation of scikit-learn [30].

	Covariance type	Initialization alg.	Prior type	No. init.	Max. iters
GMM	full, **spher.**, diag, tied	**K-Means**, **rand.**	–	–	1,**2**,**5**
Bayesian GMM	full, **spher.**, diag, tied	**K-Means**, **rand.**	**process**, **distr.**	–	1,**2**
K-Means (LR)	–	–	–	**5**,10,15,20	**5**,10,15,20
K-Means (d.)	–	–	–	1,5,**10**	5,**10**,25

The results show that the least time-consuming LMI models are GMM and Bayesian GMM. On the other hand, the most time-consuming model is K-Means with Logistic regression due to its two-step training design. In this case, the time expenditure can be attributed mainly to the second (supervised) part of the training (Logistic regression), which does not have the advantage of the time-efficient GPU-optimized K-Means implementation. In comparison with the building costs of the supervised LMI baseline, it appears that the unsupervised models exhibit lower RAM usage in all cases.

M-index requires the least time and memory out of all the examined indexes. Its performance in terms of building costs, compared to the LMI models, can be justified primarily by the fact that M-index is a mature index with many heuristics developed over the years to improve its baseline performance, which provides it with a considerable advantage over our newly-developed index.

5.2 Tuning of Learned Models

In all of the machine learning models, we identified several hyperparameters that influence the quality of the run in a major way – we list them in Table 2.

In Mixture models, i.e., GMM and Bayesian GMM, *Covariance type* influences the shape of the covariance matrix, and whether each cluster has its own covariance matrix, or all components share a common one. *The initialization algorithm* represents the pre-training initialization procedure. Bayesian GMM has one extra hyperparameter, *Prior type*, which influences the initial setting of the weight concentration prior. In the case of the K-Means algorithms, we considered different *Numbers of initializations*, where we let the algorithm run multiple times with different initialization seeds to avoid stoppage in local optimum.

We have conducted more than a hundred trials with different combinations of data-sets and hyperparameter values. The best performing parameter setups per model were selected for experimental evaluation. We have chosen the best-performing setups to be the ones that achieve 90% recall for the lowest possible stop-condition, in the shortest searching time.

5.3 Results

Four unsupervised machine learning algorithms were selected, as described in Sect. 4.3 to test the capabilities of an unsupervised approach experimentally.

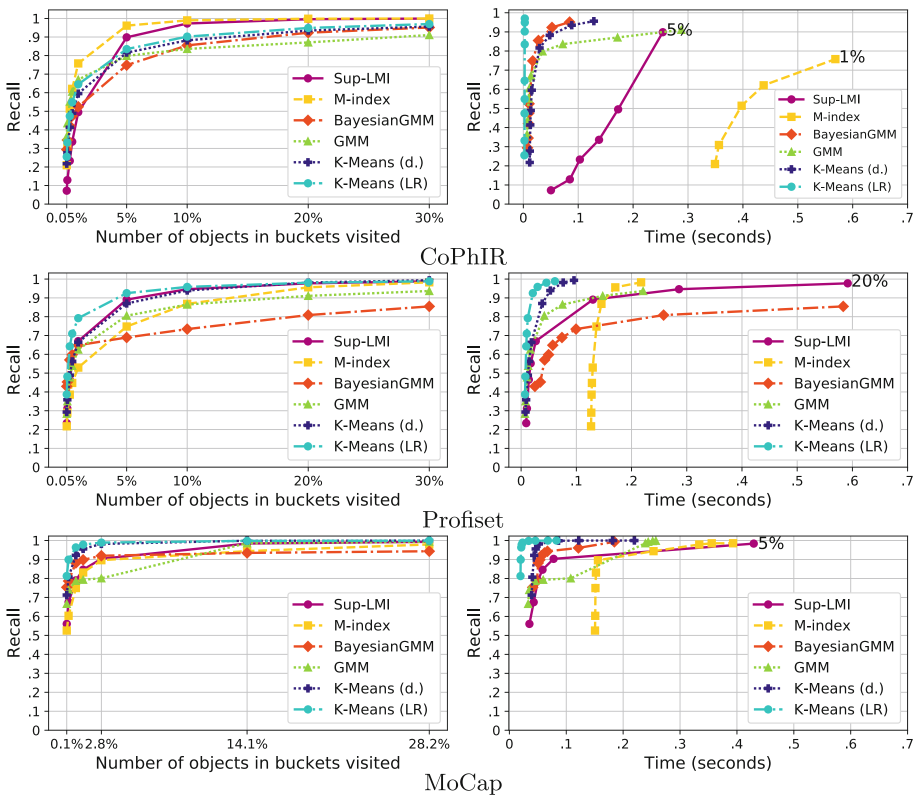

Fig. 4. Comparison between the recall of unsupervised LMI models, the best supervised setup from Antol et al. [2] (Sup-LMI), and M-index. The X-axis of the graphs on the left spans 30% of the total index size.

Two of them (*GMM*, *BayesianGMM*) represent a standard application of LMI unsupervised training and searching algorithm for data without labels. The third (*K-Means (d.)*) is constructed using unsupervised clustering combined with standard distance-based searching. The final one (*K-Means (LR)*) involves a training approach that combines unsupervised clustering and supervised learning using logistic regression. Figure 4 displays the achieved recall using two measures – the percentage of the structure searched and the time needed to evaluate one query. We compare the results of unsupervised LMI with two benchmarks: the best-performing M-index[3] and the supervised LMI[4] from our previous work [2].[5]

[3] The configurations of M-index selected as baselines for our three data-sets [2]: *M-index CoPhIR 200*, *M-index Profiset 2000* and *M-index MoCap 2000*.

[4] Best LMI setups in [2]: Multi-label trained on CoPhIR (M-index 200), Logistic Reg. trained on Profiset (M-tree 2000) and Neural net. trained on MoCap (M-index 2000).

[5] The best performing setup was the one achieving 90% recall in the lowest stop-condition and in the shortest time.

Our experiments show that in terms of navigation efficiency (i.e., recall per number of visited objects – left column), unsupervised indexes fall behind in the case of CoPhIR, but dominate both baselines in Profiset and MoCap. We attribute the poor performance seen in CoPhIR to two factors: the length of descriptors and their origin. CoPhIR's descriptors are composed of hand-picked features of the images, such as color histogram, whereas Profiset and MoCap's descriptors are extracted from machine learning models. Unsupervised LMI exhibits a better ability to traverse the indexing structure in case of more complex descriptors of machine-learning origins. In this type of descriptors, the average gain in recall over the CoPhIR data-set ranges from 4.5% to 13.5% (given the 5% stop-condition), depending on the algorithm used.

The performance difference is much more decisive when comparing time efficiency. Both of the baselines fall behind the unsupervised LMI setups significantly in all three data-sets (see the right column).[6] Specifically, the K-Means unsupervised methods reach 90% recall faster than M-index by a factor of approximately 70 (e.g., 0.02 s vs 1.55 s on MoCap) and faster than supervised LMI by a factor of approximately 8 (e.g., 0.02 s vs 0.123 s on Profiset).

K-Means with Logistic Regression is reaching the highest recall and the shortest search time. The index is able to find the relevant objects very quickly, achieving 90% recall in under 20 ms in every data-set. **Distance-based K-Means** also exhibits a favourable recall-to-speed trade-off on all of the data-sets, with performance similar to K-Means trained with the two-step approach. This setup also outperforms both of the baselines throughout all stop-conditions on Profiset and MoCap. **Mixture models** – GMM and BayesianGMM – generally show worse performance than K-Means-based indexes, and they are only competitive within the CoPhIR data-set. In most instances, mixture models only manage to outperform the baselines in the lower stop-conditions ($\leq$5%). However, they stay close behind in the higher stop-conditions in the case of Profiset and MoCap (except for BayesianGMM in Profiset).

5.4 Summary

We consider results of our experiments to be very encouraging. In the overwhelming majority of stop-conditions, both of the K-means-based setups were able to outperform M-index, as well as the best LMI setups from [2]. While this is true for both of our performance metrics – recall per number of objects searched and recall per time – the advantage of our unsupervised setups is much more prominent when considering the time-based metric.

The performance of distance-based K-Means demonstrates that the concept of LMI can be extended to work with distances instead of probabilities, with no degradation in performance. Out of all the indexes, the two-step training method achieved the most promising searching speeds and the highest recall per percentage of the structure searched in every stop-condition.

[6] For the sake of consistency of the environments across indexes, we used the Python 3.6 implementation of M-index from [2].

The GMM-based indexes perform better than the baselines on lower stop-conditions, but the performance gain disappears later in the search. As a result, these indexes still might be preferred in scenarios where one is limited by the time or the amount of the structure that can be searched, and tolerates lower recall, possibly in exchange for more favourable building costs.

6 Conclusion

In this paper, we extend the capabilities of the Learned Metric Index – a novel, machine-learning-based indexing paradigm introduced in [2]. We present a new means of LMI construction that builds the index from scratch – no pre-existing index is needed to guide the building process. Our experiments confirm that building an unsupervised LMI is a viable approach, and clustering algorithms within LMI create meaningful divisions of the data. In comparison to the formerly introduced supervised LMI, the building costs are significantly lower. By far the most significant benefit of unsupervised LMI is the overall search performance measured as recall in time – our new approach managed to beat both benchmarks (M-index and supervised LMI) by at least one order of magnitude in all cases. If we measured performance as recall per portion of the index structure visited (navigation), unsupervised LMI was superior to both benchmarks by approximately 10% in two out of the three tested data-sets. On the third data-set, the unsupervised methods fell behind when searching a larger portion of the structures. However, even in these cases, the computation speed of the unsupervised LMI outweighs the navigation deficit and reaches all accuracy thresholds in shorter time.

The performance of unsupervised LMI shown in our experiments invites for future research. This work has demonstrated the architecture of unsupervised LMI in a typical domain where similarity is obtained from vectors. These vectors are extracted directly from the objects' raw data, which is an ideal scenario for standard machine learning models. However, other types of complex data, e.g., protein structures, use different concepts of similarity – this means that their processing by LMI may not be so straightforward. In these domains, we need to employ more specialized machine learning models, such as LSTM [9], Transformer [32], or Word2vec [20] to produce vector data.

Furthermore, there is room for improvement in decreasing the construction costs by exploring different libraries and environments for the building of LMI. We also plan to inspect other machine learning models to improve LMI's pattern recognition potential even further. Finally, we plan to explore the topics of index dynamicity (i.e., the ability to locate objects outside of the indexed data-set), priority queue optimization, testing the LMI on different data-sets from different domains, and finding suitable hardware setups for LMI operations.

Overall, we view this work as an additional proof that the adoption of machine learning techniques in similarity searching is worth deep exploration, and that the concept of Learned Metric Index can provide significantly better results when it is built without a pre-existing traditional index as a template.

References

1. Antol, M., Dohnal, V.: BM-index: balanced metric space index based on weighted Voronoi partitioning. In: Welzer, T., Eder, J., Podgorelec, V., Kamišalić Latifić, A. (eds.) ADBIS 2019. LNCS, vol. 11695, pp. 337–353. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28730-6_21
2. Antol, M., Ol'ha, J., Slanináková, T., Dohnal, V.: Learned metric index — proposition of learned indexing for unstructured data. Inf. Syst. **100**, 101774 (2021)
3. Batko, M., et al.: Building a web-scale image similarity search system. Multimedia Tools Appl. **47**(3), 599–629 (2009)
4. Berrendorf, M., Borutta, F., Kröger, P.: k-distance approximation for memory-efficient RkNN retrieval. In: Amato, G., Gennaro, C., Oria, V., Radovanović, M. (eds.) SISAP 2019. LNCS, vol. 11807, pp. 57–71. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32047-8_6
5. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. (CSUR 2001) **33**(3), 273–321 (2001)
6. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search in metric spaces. In: Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 1997), Athens, Greece, 25–29 August 1997, pp. 426–435. Morgan Kaufmann (1997)
7. Dong, Y., Indyk, P., Razenshteyn, I.P., Wagner, T.: Learning space partitions for nearest neighbor search. In: 8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, 26–30 April 2020 (2020)
8. Ferragina, P., Vinciguerra, G.: The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. Proc. VLDB Endow. **13**(8), 1162–1175 (2020)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
10. Houle, M.E., Nett, M.: Rank cover trees for nearest neighbor search. In: Brisaboa, N., Pedreira, O., Zezula, P. (eds.) SISAP 2013. LNCS, vol. 8199, pp. 16–29. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41062-8_3
11. Hünemörder, M., Kröger, P., Renz, M.: Towards a learned index structure for approximate nearest neighbor search query processing. In: Reyes, N., et al. (eds.) SISAP 2021. LNCS 13058, pp. 95–103 (2021)
12. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with GPUs. arXiv preprint arXiv:1702.08734 (2017)
13. Lin, K.-I., Yang, C.: The ANN-tree: an index for efficient approximate nearest neighbor search. In: Proceedings Seventh International Conference on Database Systems for Advanced Applications, DASFAA 2001, pp. 174–181, April 2001
14. Kraska, T., Beutel, A., Chi, E.H., Dean, J., Polyzotis, N.: The case for learned index structures. In: Proceedings of the 2018 International Conference on Management of Data, SIGMOD 2018, pp. 489–504. Association for Computing Machinery (2018)
15. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. **25**, 1097–1105 (2012)
16. Li, W., et al.: Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. IEEE Trans. Knowl. Data Eng. **32**(8), 1475–1488 (2020)
17. Llaveshi, A., Sirin, U., Ailamaki, A., West, R.: Accelerating B+tree search by using simple machine learning techniques. In: AIDB — VLDB Workshop on Applied AI for Database Systems and Applications (2019)

18. Macke, S., et al.: Lifting the curse of multidimensional data with learned existence indexes. In: Workshop on ML for Systems at NeurIPS, pp. 1–6 (2018)
19. Mic, V., Novak, D., Zezula, P.: Binary sketches for secondary filtering. ACM Trans. Inf. Syst. **37**(1), 1:1–1:28 (2019). https://doi.org/10.1145/3231936
20. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
21. Moriyama, A., Rodrigues, L.S., Scabora, L.C., Cazzolato, M.T., Traina, A.J.M., Traina, C.: VD-tree: how to build an efficient and fit metric access method using Voronoi diagrams. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC), p. 327–335. ACM, New York (2021)
22. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: International Conference on Computer Vision Theory and Applications (VISAPP), pp. 331–340 (2009)
23. Müller, M., Röder, T., Clausen, M., Eberhardt, B., Krüger, B., Weber, A.: Documentation Mocap database HDM05. Technical report, CG-2007-2, Universität Bonn (2007)
24. Nathan, V., Ding, J., Alizadeh, M., Kraska, T.: Learning multi-dimensional indexes. In: Proceedings of the 2020 International Conference on Management of Data (SIGMOD), pp. 985–1000. ACM (2020)
25. Navarro, G., Reyes, N.: Dynamic spatial approximation trees. J. Exp. Algorithmics **12** (2008). https://doi.org/10.1145/1227161.1322337
26. Novak, D., Batko, M., Zezula, P.: Metric index: an efficient and scalable solution for precise and approximate similarity search. Inf. Syst. **36**, 721–733 (2011)
27. Novak, D., Batko, M., Zezula, P.: Large-scale image retrieval using neural net descriptors. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1039–1040. ACM (2015)
28. Novak, D., Zezula, P.: Rank aggregation of candidate sets for efficient similarity search. In: Decker, H., Lhotská, L., Link, S., Spies, M., Wagner, R.R. (eds.) DEXA 2014. LNCS, vol. 8645, pp. 42–58. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10085-2_4
29. Oosterhuis, H., Culpepper, J.S., de Rijke, M.: The potential of learned index structures for index compression. In: Proceedings of the 23rd Australasian Document Computing Symposium (ADCS) (2018). https://doi.org/10.1145/3291992.3291993
30. Pedregosa, F., et al.: Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
31. Sablayrolles, A., Douze, M., Schmid, C., Jégou, H.: Spreading vectors for similarity search. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019. OpenReview.net (2019)
32. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
33. Wang, H., Fu, X., Xu, J., Lu, H.: Learned index for spatial queries. In: 20th IEEE International Conference on Mobile Data Management (MDM), pp. 569–574 (2019)
34. Xiang, W., Zhang, H., Cui, R., Chu, X., Li, K., Zhou, W.: Pavo: a RNN-based learned inverted index, supervised or unsupervised? IEEE Access **7**, 293–303 (2019)