A meta-learning configuration framework for graph-based similarity search indexes

Rafael S. Oyamada [a,*], Larissa C. Shimomura [b], Sylvio Barbon Jr. [c], Daniel S. Kaster [d]

[a] University of Milan, Milan, Italy
[b] Eindhoven University of Technology, Eindhoven, Netherlands
[c] University of Trieste, Trieste, Italy
[d] University of Londrina, Londrina, Parana, Brazil

A B S T R A C T

Similarity searches retrieve elements in a dataset with similar characteristics to the input query element. Recent works show that graph-based methods have outperformed others in the literature, such as tree-based and hash-based methods. However, graphs are highly parameter-sensitive for indexing and searching, which usually demands extra time for finding a suitable trade-off for specific user requirements. Current approaches to select parameters rely on observing published experimental results or Grid Search procedures. While the former has no guarantees that good settings for a dataset will also perform well on a different one, the latter is computationally expensive and limited to a small range of values. In this work, we propose a meta-learning-based recommender framework capable of providing a suitable graph configuration according to the characteristics of the input dataset. We present two instantiations of the framework: a global instantiation that uses the whole meta-database to train meta-models and a dataset-similarity-based instantiation that relies on clustering to generate meta-models tailored to datasets with similar characteristics. We also developed generic and tuned versions of the instantiations. The generic versions can satisfy user requirements in orders of magnitude faster than the traditional Grid Search. The tuned versions provide more accurate predictions at a higher cost. Our results show that the tuned methods outperform the Grid Search for most cases, providing recommendations close to the optimal one and being a suitable alternative, particularly for more challenging datasets.

1. Introduction

Complex data (images, long texts, audios, etc.) are commonly used in pattern recognition, image retrieval, data mining, and other tasks. In general, complex data are represented through feature vectors. Such vectors are composed of measures and properties extracted from the intrinsic content of the data and retrieved using dissimilarity relations between pairs of feature vectors [1]. Complex data retrieval relies on the so-called *similarity queries*, which retrieve the dataset elements that satisfy a given similarity-based criterion. The most popular similarity query is the *k*-Nearest Neighbor query (*k-NNq*), which selects the *k* most similar elements to the query element [2].

In the literature, there are several access methods suitable for indexing complex data. These methods can be divided into four groups: tree-based [3–5], hashing-based [6–10], permutation-based [11–15], and *graph-based* [2,16–22]. The methods in each group can also be classified according to other features such as dynamicity (static or dynamic), storage type (memory or disk), and query answer exactness (exact or approximate). With the advance of big data, new distributed access methods were also proposed. Some of them are M-CAN [23] and M-Grid [24]. These methods exploit a grid-like infrastructure to organize the objects according to the similarity in different network nodes, making it possible to run similarity queries on this infrastructure. In this article, we focus on graph-based methods as recent works have shown that this method type has often outperformed other method types in approximate similarity search [13,19,25].

Widely used graph-based methods, such as the *k-NNG* [18] and the *NSW* [19], are highly sensitive to user-defined parameters for both construction and querying. The main construction parameter for these graphs is the number of neighbors an element (vertex) should be connected to. A large value adds more edges to the graph, generating shorter paths for the query algorithms to traverse towards the most similar elements to the query element. However, many neighbors also increase the memory footprint

* Corresponding author.
E-mail addresses: rafael.oyamada@unimi.it (R.S. Oyamada), l.capobianco.shimomura@tue.nl (L.C. Shimomura), sylvio.barbonjunior@units.it (S. Barbon Jr.), dskaster@uel.br (D.S. Kaster).

and the cost of vertex expansion during the search, as it is proportional to the size of the vertex's adjacency. Additionally, the query algorithm may not find a path from the search starting vertex to every vertex that is part of the answer depending on the graph structure. A common approach to alleviate this problem is to execute a given number of traversals, starting each execution from a different vertex. The number of traversals, or restarts, is also a sensible parameter as it allows to improve the result accuracy at the cost of degrading the execution time. Setting suitable values for these parameters is challenging as the parameters depend on several factors, including the type of graph-based method, the dataset properties, and the optimization goal (e.g., query time or memory requirements).

Recent works showed that there are no default parameters for widely used graph-based methods [22,26]. Aumüller et al. [26] proposed a benchmark to understand better Approximate Nearest Neighbor (ANN) algorithms, including graph-based methods. Nevertheless, the different datasets' experimental results do not reveal a parameterization that performs well for every case. We also performed a deep behavior analysis of the main graph-based methods given their settings, regarding several metrics, such as search and construction time, recall, and memory usage [22]. Our results indicated that neither a graph type is the best for all cases nor a universally suitable parameterization exists for any graph type. The results also showed that there are trade-offs between construction and search algorithms according to their parameters. However, no related work addressed the problem of assisted parameterization for these methods.

This article presents a novel intelligent approach for recommending a suitable graph-based method and its parameters for a given dataset. The core contribution is based on the premise that it is possible to learn the relation between dataset characteristics and indexing method parameters and search performance achieved by nearest neighbor algorithms using a knowledge base of similarity retrieval experimental results. Thus, it is possible to predict how a search operation through a particular index configuration would perform on a dataset and provide a proper recommendation for the case.

We design our solution based on meta-learning techniques, focusing on similarity searching on image databases and three of the most widely used base graph-based methods to execute approximate queries. Our proposed meta-models relate dataset characteristics, graph-based methods, query and indexing parameters, and search performance (e.g., execution time and query accuracy). The recommendation is the graph-based method type and the construction and query parameters expected to achieve the requested query accuracy in the shortest time or consume the smallest memory for the given dataset. We also proposed tuning methods, which rely on sample experiments on the new dataset, defined by a grid search over a small range of parameters, to enhance the training set and generate meta-models tuned to the dataset.

In a previous version of this work [27], we developed a recommender instantiation based on a global meta-model learned on the whole repository of dataset instances collected. This work introduces a novel instantiation by learning a set of meta-models from clusters of similar datasets. We noticed that separate models for different classes of datasets provide superior prediction performance than a global model without losing generality. We called this strategy the *dataset-similarity-based* meta-learning recommendation, which consists of selecting the characteristics of a dataset that most impact the graphs' performances to measure the similarity among datasets, identifying clusters of similar datasets, and generating meta-models adjusted to each cluster. The meta-models provide highly precise predictions for new datasets with properties similar to the properties of the datasets in the cluster. The graph-based index recommendation for a new dataset finds the cluster closest to the dataset and selects the meta-models from the cluster for inferencing. The work also extends our previous approach by expanding the dataset characterization with additional measures related to the dataset hardness for the nearest neighbor problem and applying a new data augmentation technique based on interpolation.

We summarize our contributions as follows.

- Proposal of a general framework based on meta-learning for parameterizing graph-based methods for similarity searches.
- Proposal of a variety of dataset descriptors for similarity searching purposes.
- Proposal of the global and the dataset-similarity-guided meta-learning instantiation strategies capable of finding graph configurations that satisfy the user requirements more precisely or faster than commonly employed strategies.
- Implementation of recommenders following the proposed instantiation strategies, including variations that employ fine-tuning of the meta-models using instances extracted from the input dataset at recommendation time.
- Extensive experimental evaluation analyzing the behavior of recommenders' main building blocks, comparing them with baselines and our global meta-model instantiation [27], and providing details about the development of our proposals.

The dataset-similarity-based recommender instantiation outperformed the global instantiation and other two baselines: (a) the best general setup, i.e., the parameterization that achieves the best performance for most datasets, and (b) the grid search procedure, which is the most adopted method for parameter setting. In general, the recommendations are comparable to those provided by the grid search approach but demand orders of magnitude less recommendation time as grid search requires building sample graph-based indexes and executing queries on them. Nevertheless, the tuned instantiations, which are also costly, outperform the grid search in most cases, often providing recommendations close to the optimal one.

This article is organized as follows. Section 2 presents the background to understanding our proposal, including an overview of similarity searches, focusing on proximity graphs, the main concepts of meta-learning, metrics that measure how complex a specific similarity search problem is. Section 3 discusses this work's research problem, exemplifying the impact of parameters for graph-based methods, and presents related work. Section 4 describes our proposed framework to recommend graph configurations for similarity searching, modeled as a meta-learning problem. Subsequently, Section 5 describes the two instantiation strategies of our framework: the global and the dataset-similarity-based meta-learning approaches. Section 6 shows an extensive experimental evaluation of the instantiation strategies, and Section 7 presents an analysis of the supporting techniques of our proposal. Finally, we conclude this work and mention further research in Section 8.

2. Background

2.1. Proximity graphs for similarity search

Similarity has been the basis of the management and retrieval of complex data as these types of data do not follow a total order relation to be compared through standard operators such as $<$, $>$, $\leq$, $\geq$. In the context of this work, complex data refer to data containing rich content, such as images, videos, and audios, which are usually represented by feature vectors extracted from the data content. Feature vectors allow users to measure the similarity between pairs of complex data.

The two basic types of similarity queries are the *Range* query (*Rq*) and the *k*-Nearest Neighbor query (*k-NNq*) [1]. The *Range* query is defined as $Rq(\delta, s_q, \xi) = \{s_i \in S | \delta(s_q, s_i) \leq \xi\}$, where $S \subseteq \mathbb{S}$ is a dataset in complex data domain $\mathbb{S}$, $s_q \in \mathbb{S}$ is a reference (or query) element, δ is a distance function defined in $\mathbb{S}$, and $\xi \in \mathbb{R}_+$ is a distance threshold. This query retrieves all elements $s_i \in S$ whose distance to the reference element s_q is less than or equal to ξ. On the other hand, the *k-NNq* takes a given query element $q \in \mathbb{S}$ and returns the k closest elements to q from $S \subseteq \mathbb{S}$ according to a distance function δ. The retrieved set $S_k \subset S$ can be expressed as $kNNq(\delta, s_q, k) = S_k = \{s_i \in S \mid |S_k| = k, \ \forall s_j \in S - S_k, \delta(s_q, s_i) \leq \delta(s_q, s_j)\}$.

A proximity graph is a data structure capable of performing similarity queries. It can be defined as a graph $G = (V, E)$ in which each pair of vertices $(v, u) \in V$ is connected by an edge $e = (u, v)$, $e \in E$, if and only if u and v satisfy a given property P, called neighborhood criterion. The neighborhood criterion defines the type of the proximity graph. Such a property is usually the dissimilarity between a pair of vertices, computed by a distance function $\delta(u, v)$. Among the types of proximity graphs for similarity searches, the *k*-Nearest Neighbor Graph (*k-NNG*) [25,28] and the Navigable Small-World graph (*NSW*) [19] are two fundamental types of graphs.

The *k-NNG* [29] is defined as a graph $G = (V, E)$, where $E = \{(u, v), v \in NN_k(u)_\delta\}$, being $NN_k(u)_\delta$ the set containing the k nearest neighbors of u in the set of vertices V according to the similarity function δ. Its edges can be directed or undirected. If the *k-NNG* is weighted, the weights usually express the distance between the connected pairs of vertices ($\delta(u, v)$). The *k-NNG* has well-known properties useful for performing similarity searches, therefore, it has been used as the base for several other graph-based methods. Examples include methods that build approximate versions of the *k-NNG* at a lower cost than the exact construction [30] and methods that relax the connectivity of every element to its *k*-nearest neighbor [31]. The brute-force construction of the *k-NNG* has a quadratic computational cost, which is unfeasible for large datasets [28]. A remarkable method that generates an approximated version of a *k-NNG* is the *NN-Descent* [30]. The main supporting idea of the *NN-Descent* is that "the neighbor of a neighbor is probably a neighbor". This idea guides the graph's construction, enabling a considerable reduction in the execution time for many cases. It starts by computing a random approximation of the nearest neighbors of each element $v \in V$. Subsequently, it iteratively improves this approximation by comparing each element to the neighbors of its neighbors, including both *k-NN* and reverse *k-NN*. The construction finishes when the process ceases to show improvement on the results from the previous iteration.

The *Navigable Small World* (*NSW*) graph is also based on connecting elements to their nearest neighbors. However, it uses short- and long-range undirected edges that grant the graph small-world properties [19]. Small-world properties were also considered in the experimental design of self-organizing search systems [32]. The main advantages of the *NSW* are (i) fast and highly precise approximate search execution, thanks to the small-world properties, and (ii) fast construction algorithm. The *NSW* has two types of edges: the regular, or short-range, links, and the long–short links, responsible for the navigable small-world properties [33]. The construction procedure inserts incoming elements iteratively as vertices in the graph. A new vertex is connected to a set that contains its closest neighbors in the graph according to an approximate *k-NN* search algorithm. New edges are short-range links because they connect the vertex to its k nearest neighbors according to the current state of the graph. Along the insertion process, the short-range edges tend to become long-range edges because new vertices become the new closest neighbors to previously inserted vertices, increasing their adjacency.

There are several other types of proximity graphs in literature [7,16,20]. However, the *k-NNG* and the *NSW* have been used as bases for the current state-of-the-art methods [34]. Therefore, we employed these base methods for the development of this work.

2.2. Searching in proximity graphs

The spatial approximation, introduced by Navarro [2], has become a fundamental approach for similarity searching in proximity graphs. Given a query object q and a proximity measure δ, the spatial approximation starts the search from a source vertex $u \in V$, and iteratively traverses the graph using greedy steps to get spatially closer and closer to the elements that are the most similar to the query element q regarding δ. At each step, the search is propagated from a vertex u to its neighbors $N(u)$ that are closer to q then u.

The spatial approximation property allows exact answers to similarity queries on proximity graphs under certain conditions. Given a metric space [1] $\langle S, \delta \rangle$ and a graph $G = (V, E)$, where $V \subseteq S$ and every $e \in E$ has the form $e = (u, v)$ such that $v \in N(u)$, G must fulfill the property shown in Eq. (1) to correctly answer similarity queries using a search algorithm based on the spatial approximation approach, for any query object $q \in S$.

$$\forall u \in V, \ \textit{if} \ \forall u \in N(u), \delta(q, u) \leq \delta(q, v), \\ \textit{then} \ \forall v' \in V, \delta(q, u) \leq (q, v') \tag{1}$$

The spatial approximation has been employed in several works in the literature to produce approximate answers to similarity queries, as exact answers are too expensive. Approximate similarity searching reduces the search time, considering that a given answer may not be exact but close enough to the exact one to be helpful. Thus, many works have concentrated on finding alternatives to decrease the search time with the minimum error possible.

A common similarity search approach on proximity graphs is selecting initial vertices and using a best-first search process based on the spatial approximation. This approach produces approximate results for most proximity graphs. The Graph Nearest Neighbor Search (*GNNS*) [25] is an effective algorithm for the so-called Approximate Nearest Neighbor (ANN) search. The *GNNS* can be considered a seminal algorithm as other proposals employ similar ideas. The algorithm executes multiple greedy searches based on spatial approximation and aggregates the partial results into the final result. The multiple searches are called *restarts* (R), whose number is a user-defined parameter. The R parameter allows improving the quality of the result as each search starts from a different source and traverses a different path in the graph. Nevertheless, the number of restarts also impacts query execution time. Setting R and other parameters for graph-based methods is challenging, as we discuss later in Section 3.

2.3. Meta-learning

Meta-learning differs from the traditional view of learning (a.k.a. base-learning) in the notion of what to learn [35]. Traditional learning induces a predictive function for a single problem domain, while meta-learning attempts to gather knowledge about one or more learners applied to several domains. In short, this approach learns how to learn across several domains. Meta-learning has been used for different purposes, such as prediction of algorithm performance [36], prediction of the training runtime [37,38], decision if an algorithm should be tuned or not [39], recommendation of hyperparameters [40,41], ranking algorithms [42], and so on.

The accumulated knowledge from problems or tasks is usually called meta-data or meta-knowledge. The meta-knowledge comprises several components, as follows.

- *Meta-features* (a.k.a. descriptors or characterization measures): features to characterize datasets. For instance, the cardinality and dimensionality.
- *Meta-targets*: targets to be predicted, which may be the performance obtained by algorithms in the different domains where they were applied (this is the most relevant definition for this work; however, there are other types of meta-targets [43]).
- *Meta-instances*: the junction of meta-features with their corresponding meta-targets.
- *Meta-dataset*: a set composed of meta-instances.
- *Meta-models*: learning algorithms that map meta-features to the meta-targets.

We can formally define meta-learning as follows. Consider a repository of datasets D (or tasks), a dataset $d_i \in D$, and an algorithm configuration θ_j in the configuration space Θ. Consider also a set of performance evaluations P, where $P_{ij} = P(d_i, \theta_j)$ is an evaluation metric, e.g. accuracy, measured by executing the base-learner/base-method with the configuration θ_j on the dataset d_i. Each dataset is described by a vector of meta-features $m_i \in M$, being $|D| = |M|$. Therefore, $M \cup P$ forms a meta-dataset. Meta-learning is the act of applying a meta-learner to induce a meta-model to find patterns and measure similarities on the meta-dataset. Depending on the induction strategy, the final meta-model can be used to predict the performance of the base-learners/base-methods considered or to recommend a suitable configuration θ_j for a new dataset d_{new}.

2.4. Measuring the hardness of similarity searching

A subject that has received attention in recent works is the analysis of which properties of a dataset make it more challenging than others for similarity retrieval, including the ANN queries, which we address in this work. The hardness of the dataset is a useful feature to define suitable parameters for indexes for similarity retrieval. Researchers have explored properties such as the intrinsic dimensionality [44–46], and the concentration phenomenon [47–49].

Finding the actual nearest neighbor is considered difficult when the space is high-dimensional, and this phenomenon is well known as the "curse of dimensionality" [44]. However, although a dataset is in a high-dimensional space, it can usually be embedded into a space with fewer dimensions with little or no information loss. The *intrinsic dimensionality* (ID) can be seen as the minimum number of latent variables to describe the data. It aims at finding the dimensionality of a surface that best approximates the original data minimizing the loss of information [50].

The literature provides several methods for estimating the ID of a dataset. For instance, it can be estimated through the PCA method, where the number of principal components that explain a certain variance of the data represents the ID [51]. Levina and Bickel [50] also proposed another method based on the maximum likelihood estimation.

Likewise, the phenomenon of the concentration of distances is another property that can be employed to measure the dataset complexity. The concentration of distances arises when all pairwise distance between points falls within a small range, which means that the nearest and farthest neighbors of elements are similar. This phenomenon often occurs in high-dimensional spaces. Regarding similarity retrieval, the concentration of distances deeply affects the pruning ability of indexing methods, often leading them to present linear cost [52–54]. To estimate how concentrated a dataset is, François et al. [49] proposed to measure the Relative Variance (RV) of the norm. The RV is a non-negative rate, and small values indicate a high concentration of distances. Intuitively, it estimates the concentration by relating a measure of spread (variance) to a location (expectation) measure.

3. The graph-based method parameterization problem and related work

We show in this section that defining suitable values for these parameters has a major impact on the effectiveness and performance of the methods and is a challenging problem. Both the *k-NNG* and the *NSW* are sensible to construction parameters, particularly the number of neighbors of each vertex (*NN*), which defines the number of edges in the graph. Choosing the correct *NN* parameter is essential for an optimal trade-off, as it directly impacts construction time, memory usage, query answer quality, and query execution time. A low value for *NN* results in low memory usage, as the fewer edges a graph has, the less memory it needs to be stored, and low construction time because of the search process in vertices' insertion. However, low *NN* values also lead to poor-quality queries in most cases.

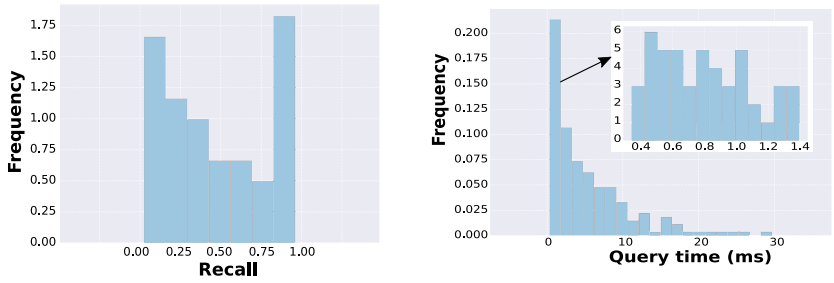
Query algorithms often have parameters to be tuned as well. Notice that we use *NN* to define the construction parameter number of neighbors of the graph-based methods and *k* to define the query parameter number of neighbors in a *k-NN* query. For instance, the *GNNS* algorithm depends on the *R* parameter. A low *R* parameter is desirable since it implies faster query time execution. However, the quality of the result is degraded if the value is too low.

Our discussion considers typical parameterization scenarios. After that, we discuss related work aimed at addressing this problem.

3.1. Typical parameterization scenarios

The first scenario is when the user makes a careless choice and uses the same configuration across different datasets. This scenario usually happens when a configuration provides a good result for a given dataset. Then, for simplicity, the user replicates the "good" configuration for other datasets. To illustrate this scenario, we fixed the configuration, built a graph-based method for different datasets, executed *k-NN* queries using these indexes, and analyzed the results. We ran this test using several configurations varying the graph type and the construction and query parameters. Fig. 1(a) shows results of a representative example, which corresponds to a *NN-Descent* graph set with $NN = 25$ running *k-NN* queries with $k = 30$ using the *GNNS* algorithm with $R = 10$ for all datasets used in this work (see details in Section 6.1). The figure shows the distribution of the average recall rates throughout all datasets, being the recall rate for each dataset computed as the average recall for 100 queries with query elements randomly chosen from the dataset. The query elements are not included in the constructed graph. The recall rate of a query is the fraction of the true *k*-nearest neighbors to the query element retrieved by the query. It is noticeable that the same set of parameters for distinct datasets leads to entirely different quality rates for queries. Similar reasoning is also valid regarding query time.

The second scenario considers a single dataset. In this scenario, the user has to set the ideal parameters for the dataset subject to some constraints. Fig. 1(b) presents the distribution of the average execution time for 30-*NN* queries using the *GNNS* search in the *NN-Descent*, considering different parameters, for the dataset *Color Histogram*, whose features are the 32-bin color histogram of a set of 68,040 images. Although the real problem also requires selecting the best graph-based method for the case, the goal here is only to define the parameters *NN* and *R* for the *NN-Descent* for this dataset. The constraint is that the query should have a recall of at least 0.95. Analyzing the query time distribution, we notice that almost two-thirds of the tested combinations of *NN* and *R* do not lie in the first bucket, which means that the



(a) (b)

Fig. 1. (a) Distribution of the recall rates for *k-NN* queries using a *NN-Descent* graph with a fixed configuration for a set of distinct datasets. (b) Distribution of the query time for varying configurations of the *NN-Descent* for *k-NN* queries with recall >0.95 using the Color Histogram dataset. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

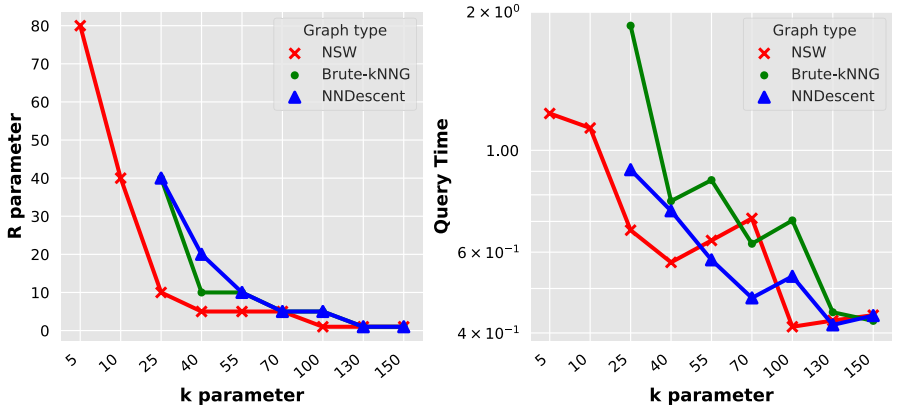


Fig. 2. The behavior of graph-based methods for the dataset Texture and increasing *NN*. (a) Smallest number of restarts for each graph and *NN* value. (b) Query time for the corresponding configurations of the plot on the left.

execution time is at least twice larger than the time demanded by the best configurations. The figure also shows the histogram of the configurations that lie in the first bucket. We can observe that the variance regarding the average execution time is also notable for the best configurations. Additionally, regarding only the top-15 configurations, the methods present a variation of up to 50% in the execution time, which is significant.

The third scenario is the complete one, as it requires choosing the best graph type and its configuration for a given dataset. Fig. 2 shows how the graph types *NSW* and *k-NNG* behave for increasing values for the *NN* parameter for the dataset Texture, which has texture features of 68,040 images. The figure shows the results for the *k-NNG* built using two construction algorithms: *NN-Descent* and brute-force (*Brute-kNNG*). Each point in the plots corresponds to the smallest number of restarts and, consequently, the shortest query time for the corresponding type of graph and *NN* value that returned results subject to the constraint of having a recall of at least 0.95. Analyzing the plots, if the optimization goal is memory (i.e., the configuration that satisfies the constraint with the least memory consumption), the best option is the *NSW* with $NN = 5$. On the other hand, if the optimization goal is query time (i.e., the fastest configuration that satisfies the constraint), we have two configurations that tie: *NSW* with $NN = 100$, and *NN-Descent* with $NN = 130$, being the latter the best cost-benefit option as it demands less memory than the former one. We can also see that choosing the graph type that is the fastest in general but with a poor configuration may be the worst option among the graph types. In this case, *NSW* is the fastest in general but the worse with $NN = 70$. The opposite is also true since the *Brute-kNNG* is the slowest method in general; however, it is the fastest for $NN = 150$. Finally, the plots indicate that every method has an optimal configuration, which varies for different datasets and constraints. These reasons reinforce that the problem of recommending optimal parameters to configure graph-based methods is crucial and challenging.

3.2. Related work

Although algorithm and parameter tuning, in general, is an active research topic, there are few works considering it for nearest neighbor algorithms. Works in the literature of similarity searching have defined parameters for indexes based either on user intuition or exhaustive evaluations [21,55–58]. These approaches lead to suboptimal configurations and/or are excessively time-consuming since identifying adequate parameters is a challenging problem.

Some works present extensive evaluations of access methods for similarity searches. For instance, a comprehensive experimental study was performed by Li et al. [59] on state-of-the-art ANN algorithms to provide a better understanding of their general behavior. The authors discussed some concepts for understanding why some datasets are harder than others by evaluating sixteen algorithms. On the other hand, the *ANN-Benchmark* was proposed to standardize the evaluation of ANN algorithms [26]. This benchmark enables comparing a wide range of ANN algorithms and their configurations on several real datasets. Still, the authors state the need for developing new tuning methods, which requires getting a better understanding of the impact of intrinsic properties of the datasets on the methods.

The results reported in these works show that graph-based methods present better performance than other types of methods for most datasets. Considering this claim, we made an experimental survey of graph-based methods in a previous work [22]. We

implemented six base graph types and three query algorithms in a common platform and performed an extensive evaluation on real and synthetic datasets. Our work revealed the difficulty of finding ideal trade-offs (e.g., execution time and query quality) in the main base graph types according to user requirements. Recently, Wang et al. [60] presented a similar analysis by employing thirteen graph-based methods but using their original implementations. The authors propose new guidelines and recommendations according to different scenarios from their results. Corroborating our previous work, the authors state the difficulty of finding the best graph for each type of dataset and outstand the necessity of new solutions for this problem.

To the best of our knowledge, the first method of auto-selecting a suitable algorithm configuration for similarity searching was proposed by Muja and Lowe [61]. Their goal is to minimize a cost function based on query time, indexing time, and memory usage. However, this work only approaches tree-based methods. Therefore, it is evident that selecting suitable algorithms and parameter values for performing similarity searches is not a trivial task, and this is still an open problem in the ANN community.

Recently, learning-based solutions have been proposed in several areas of databases in general [62]. Specifically for similarity searching, learned approaches have been employed for refining proximity graphs [63], optimizing approximate queries [64,65], and replacing traditional index structures [66–68]. Following this trend, in this work, we present an intelligent approach based on meta-learning techniques to recommend graph-based methods along with their parameters.

4. A meta-learning framework for proximity graph parameter recommendation

The main objective of this work is to develop an intelligent system to recommend proximity graphs for similarity query retrieval. We draw inspiration from the success of learning-based solutions for similar problems in machine learning research. More specifically, we model our solution based on meta-learning techniques. Our ultimate goal is, among a set of graph-based methods chosen in advance, to recommend the best graph-based method, along with its optimal construction and searching parameters, for a new complex dataset, satisfying specific requirements provided by the user.

This section discusses the targeted problem, highlights the impact of parameters in graph-based methods, states how we model our solution as a meta-learning problem, and outlines the proposed recommending framework. Then, the section details the proposed strategies for gathering meta-knowledge, meta-learning, and recommending.

4.1. Modeling proximity graph recommendation as a meta-learning problem

We formally model the problem of providing graph-based recommendations and their parameters for similarity retrieval as a meta-learning problem. Let D be a repository of complex data datasets, $d \in D$ a dataset and d_f be a vector of meta-features describing d. Let θ be a pair $\langle G_T, G_\theta \rangle$, where G_T is a graph-based method type and G_θ a set of configuration parameters for the method type, and $p_{d,\theta}$ a performance measurement for batches of queries executed on an index built on d using the parameterization θ.

Definition 1 (*Meta-dataset*)**.** A meta-dataset $\mathcal{D}$ is a set composed of meta-instances with the form $I = (X, y)$, where $X = \langle d_f, \theta \rangle$ and $y = p_{d,\theta}$ is the corresponding meta-target.

Definition 2 (*Meta-model*)**.** A meta-model $\hat{f}_M(X) = \hat{y}$ is a function learned by a method M trained on $\mathcal{D}$, where $\hat{y}$ is the predicted meta-target for the instance X.

We take as inputs a new dataset d_{new}, a set of user-defined constraints C, and the desired optimization goal ψ. The set C can contain constraints like the most frequent query parameters (e.g., the typical value for k for k-NN queries) and the minimum acceptable recall. The optimization goal ψ defines what to minimize: the query time or the memory consumption. The memory consumption is proportional to the number of edges in the graph. We extract the dataset features $d_{f_{new}}$ and employ the constraints C to filter the parameterizations $\Theta_C \subseteq \Theta$ that satisfy the constraints in C, where Θ is a finite domain of graph-based method parameterizations. Then, we collect a set of predictions $Y = \{\hat{y}_1, \ldots, \hat{y}_m\}$ such that each $\hat{y}_i \in Y$ is the meta-target corresponding to a meta-feature $X_i = \langle d_{f_{new}}, \theta_i \rangle$, where $\theta_i \in \Theta_C$. Finally, we select the parameterization θ_i whose prediction $\hat{y}$ satisfies the optimization goal ψ and return θ_i as the recommended parameterization for the case.

4.2. The proposed framework

To develop the solution of the proposed problem modeling, we faced three challenges. The first one is the construction of a meta-dataset that allows generating suitable recommendations. We have to select a set of complex datasets, identify meta-targets relevant to the problem, define a procedure to generate the meta-targets, and identify meta-features that describe the dataset properties with valuable clues for graph-based method parameterization. The second challenge is the definition of meta-models that provide good prediction performance. This challenge comprises selecting and trying alternative learning methods and training strategies, including using different options to compose the training set. Finally, the third challenge is to propose a method to generate the final recommendations using the trained meta-models. This challenge includes stating what user-defined constraints apply and how to satisfy each of them. Moreover, it requires defining how to achieve the optimization goal and proposing strategies to limit the solution space.

Fig. 3 summarizes the architecture of the proposed meta-learning-based recommendation framework. It illustrates the steps of gathering meta-knowledge, performing meta-learning, and generating the recommendations according to the user's inputs. The task of building the meta-dataset takes as input a repository of datasets, a set of graph-based methods, and a set of parameter values. The set of graph configurations is the combination of graph types and values for the considered parameters. For each dataset in the repository of datasets, the architecture generates meta-instances comprising meta-features extracted from the dataset (characterization) and a graph configuration. The meta-dataset is composed of meta-instances that are associated with meta-targets. A meta-target is the historical performance measurement (e.g., average query time or recall) obtained by running queries with different graph-based methods. These methods are built based on a wide range of original problems combination and query configuration values.

Subsequently, the recommender applies a meta-learner to build a meta-model base with different regressors trained using the meta-dataset. The user provides a new dataset and some preferences to obtain a recommendation, including the requirements to be satisfied (e.g., query preferences, specific meta-model) and the optimization goal (e.g., query time or memory usage). The input dataset is characterized, and the recommender generates candidates represented by a set of meta-instances that simulate different parametrizations for the given dataset. The

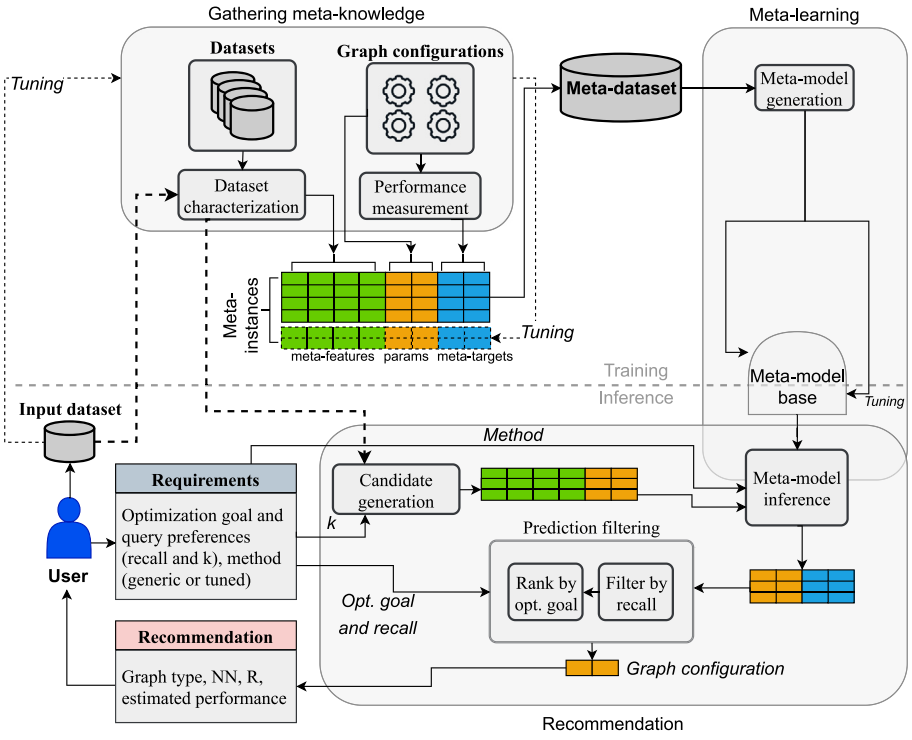


Fig. 3. The proposed framework: the process of gathering meta-knowledge to build the meta-dataset, the instantiation and usage of meta-models, and the generation of recommendations according to inputs provided by a user.

meta-models take this set of meta-instances and infer the corresponding performance measures (meta-targets) for each of them. Finally, it filters the meta-instances according to the performance measure and optimization goal and returns the user the best parameterization.

Fig. 3 also shows an alternative tuning path. This path generates a fine-tuned meta-model for the input dataset, achieving higher prediction accuracy than the generic meta-model at the cost of being substantially slower to output the recommendation. The input dataset is submitted for characterization and performance measurement on the supported graph-based methods with varying parameter settings. This process enriches the meta-dataset with meta-instances from the input dataset, allowing tuning the meta-model for the case.

Notice that even though the performance measurement may be hardware-dependent (e.g., average query time), a meta-model induced using meta-targets from this measurement should be effective for other hardware as our recommender is based on relative performance using ranking. The following sections detail the techniques we propose for gathering meta-knowledge, meta-modeling, and recommendation.

4.3. Strategy for gathering meta-knowledge

The strategy for gathering meta-knowledge is divided into four steps: (a) defining a repository of datasets and extracting meta-features from each of them, (b) establishing a set of graph-based method types and parameter values, (c) extracting meta-targets by measuring the performances of every graph-based method configuration on every dataset, and (d) performing data augmentation using interpolation.

There are a plethora of datasets of complex data available. The first decision is to select a reduced repository of datasets that are representative to several applications, or group of applications, that is feasible to deal with in terms of the needed effort to generate the meta-features and the meta-targets. In this work, we focus on the recommendation of graph-based methods for similarity retrieval on image databases. Therefore, as a starting point, we have an assortment of datasets with features extracted from images and enrich the meta-dataset with new datasets over time. We also include synthetic datasets varying properties, including dimensionality, cardinality, and data distribution (data sparsity and number of clusters) to augment the space of data properties. The synthetic datasets improve the generalization of our approach.

After collecting datasets and defining the indexing methods to support them, building the meta-database comprises the following steps. First, we perform the characterization of each dataset extracting its meta-features. Subsequently, we run batches of experiments to compute the meta-targets regarding each graph configuration. We use a weak labeling approach regarding the meta-targets to perform data augmentation. Finally, we compose the final meta-dataset by generating meta-instances in the form $I = (\langle d_f, \theta \rangle, y)$, as described in Section 4.1. In the following subsections, we define the meta-features and meta-targets we employ in our solution.

4.3.1. Meta-features

The characterization of datasets must consider the nature of the problem. As there is no work in the literature addressing meta-learning techniques for similarity retrieval, or parameterization of indexes for complex data, this section outlines the metrics we employ as meta-features in our proposal.

The employed meta-features comprise three categories: classical meta-features, general measures commonly used in similarity retrieval evaluation, and measures of the hardness of similarity searching. The classical meta-features include general, statistical, and information-theoretical measures calculated on the dataset. The general measures include variables usually explored by the similarity search community to evaluate indexing structures and searching algorithms, such as the dataset cardinality and embedding dimensionality [59,69]. The measures of the hardness

of similarity searching are related to the complexity of datasets (see Section 2.4), including local intrinsic dimensionality, relative variance, and features derived from these complexity metrics (e.g., histograms of local intrinsic dimensionality).

Although there are different estimators available in the literature for the hardness of datasets for similarity searching, little is known about mathematical differences among them. However, it is known that each estimator is based on different properties, such as fractal properties or distance concentration. Thus, we employ different methods in order to enrich the meta-dataset: the Relative Variance (RV) of the dataset, the number of Principal Components that explain at least 90% of the variance, statistical metrics from the Local Intrinsic Dimensionality (LID) data — entropy, kurtosis, median, skewness, and standard deviation, and a histogram of instances lying in ranges of LID (ten bins). The local intrinsic dimensionalities are estimated with $k = 100$ on a sample of up to 10.000 instances of the input dataset. The LID statistical metrics are measured from these values, being the mean (*lid_mean*) the intrinsic dimension. The relative variance (*rv*) was also estimated by sampling the same instances from the input dataset.

There are several tools that generate classic meta-features (i.e., general, statistical, and information-theoretical), whereas, for the complexity metrics, there are isolated open-source implementations. The implementation we developed to generate these meta-features is available online.[1] We summarize the meta-features composing the meta-dataset built in this work in Table 1.

4.3.2. Meta-targets

The meta-targets should cover a variety of measures that impact similarity retrieval. Regarding graph-based methods, a few measurements are conflicting, such as the retrieval speed and the retrieval quality of ANN queries. The main idea is that the meta-model should provide reliable estimates so that the user can evaluate existing trade-offs among several graph configurations without building an index and/or running sample queries using each configuration. Meaningful meta-targets for our purpose are as follows.

- Recall: the average fraction of correct query answers retrieved;
- Query time: the average time for executing a query;
- Number of distance computations: the average number of distance computations during a query.

Our proposal initially considers classic graph-based methods such as *Brute-kNNG*, *NN-Descent*, and *NSW*, due to their importance in literature as base algorithms for state-of-the-art methods. However, the proposal is extensible to other indexing methods. As mentioned in Section 3, these graphs have the number of nearest neighbors (*NN*) as a parameter in common, which refers to the number of elements that each element will be connected to. The *NN* parameter is related to query execution time, query quality, and memory consumption. We focus on *k-NN* queries as they are the most typical queries on graph-based methods. A usual tuning parameter for *k-NN* queries on graph-based methods is the number of traversals starting from distinct vertices to increase accuracy, as the *GNNS*'s number of restarts (*R*). Although several algorithms could be included in our proposal, this work uses the *GNNS* because it is efficient and flexible to improve the recall besides applying to all types of graphs selected. Therefore, the graph configuration attributes we include in the meta-dataset are the graph type, the indexing parameter *NN*, and the query parameters *k* and *R*.

The meta-targets are obtained by averaging the results from batches of queries performed on a set of graph-based methods built on a dataset according to predefined parameters. Ideally, the measurement of the meta-targets should be performed in a homogeneous environment to guarantee fair evaluations. However, it is also acceptable to use separate implementations if the recommendation is employed to build graph-based methods using an equivalent implementation. For this task, our proposal relies on a library containing the considered graph-based methods developed in related works. The graph configuration attributes and meta-targets complete the meta-dataset our proposal uses, as shown in Table 1.

Our framework considers that the meta-dataset might grow over time by including new datasets, either by running offline batches or online sample experiments required by the alternative tuning path. Hardware-dependent meta-targets (e.g., average query time) should be appropriately adjusted to scale the performance obtained on different hardware settings. We are aware that scaling may introduce noise in the meta-dataset. However, the benefit of enriching the meta-dataset usually overcomes the drawback of eventually adding noise.

4.3.3. Data augmentation based on interpolation

Our framework produces recommendations in the range of valid values for the parameters because we employ regressors. Therefore, the meta-models estimate the performance for any valid parameter combination produced by the candidate generation step. However, the meta-models were fed with meta-instances derived from the performance measurement, which uses a limited combination of parameters. Therefore, we noticed an improvement opportunity by augmenting the meta-dataset with meta-instances with weak labels.

The proposed data augmentation relies on interpolating the values for meta-targets using additional parameter values for each dataset. Consider a set of meta-instances regarding a given dataset *d*, a given graph type *g*, a given number of retrieved elements *k*, and its respective meta-targets. From these meta-instances, we interpolate the meta-target using additional values for the parameters *NN* and *R*.

We claim that the noise added to the meta-dataset due to our augmentation approach's weak labeling is negligible compared to the benefit of having a larger dataset for learning. We employed the quickhull algorithm [70], implemented as the *LinearNDInterpolator*[2] method from the SciPy Python Library. This method calculates the input data triangulation to generate its interpolant, and for each triangle, a linear barycentric interpolation is performed. The interpolation is smooth, and we identified insignificant errors in validation tests using actual values. We focused the interpolation on low parameter values since these regions concentrate the most notable variability for measures.

4.4. Meta-learning

The overall idea is to regress each meta-target by finding patterns and relationships between meta-features and graph configurations and their performances. Notice that it is necessary to train at least one meta-model per meta-target.

In this work, we propose different strategies to conceive and use meta-models. For each of them, we have three approaches considering fine-tuning options. The first approach, called the Generic Meta-Model (*GMM*), has no tuning and considers only the meta-instances gathered during the construction of the meta-dataset. The second one, the Tuned Meta-Model using Grid Search

[1] https://github.com/raseidi/annmf.

[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.LinearNDInterpolator.html.

Table 1
Summary of the attributes composing the meta-dataset built in this work.

Type	Name	Category	Description
Meta-features	cov	Statistical	Absolute value of the covariance of distinct dataset attribute pairs.
	eigenvalues	Statistical	Eigenvalues of covariance matrix from dataset.
	iq_range	Statistical	Interquartile range (iqr) of each attribute.
	kurtosis	Statistical	Kurtosis of each attribute.
	mad	Statistical	Median absolute deviation (mad) adjusted by a factor.
	max	Statistical	Maximum value from each attribute.
	mean	Statistical	Mean value of each attribute.
	median	Statistical	Median value from each attribute.
	min	Statistical	Minimum value from each attribute.
	nr_cor_attr	Statistical	Number of distinct highly correlated pair of attributes.
	nr_norm	Statistical	Number of attributes normally distributed based in a given method.
	nr_outliers	Statistical	Number of attributes with at least one outlier value.
	range	Statistical	Range (max–min) of each attribute.
	sd	Statistical	Standard deviation of each attribute.
	skewness	Statistical	Skewness for each attribute.
	sparsity	Statistical	Sparsity metric for each attribute.
	t_mean	Statistical	Trimmed mean of each attribute.
	var	Statistical	Variance of each attribute.
	attr_conc	Info-theory	Concentration coefficient
	attr_ent	Info-theory	Shannon's entropy for each predictive attribute.
	attr_to_inst	General	Ratio between the number of attributes.
	inst_to_attr	General	Ratio between the number of instances and attributes.
	nr_attr	General	Total number of attributes.
	nr_inst	General	Number of instances (rows) in the dataset.
	lid_entropy	Complexity	Entropy of local intrinsic dimensionalities.
	lid_hist[1-10]	Complexity	10-bin histogram containing the number of instances in ranges of local intrinsic dimensionalities.
	lid_kurtosis	Complexity	Kurtosis of local intrinsic dimensionality.
	lid_mean	Complexity	Mean of local intrinsic dimensionalities, a.k.a. intrinsic dimensionality.
	lid_median	Complexity	Median of local intrinsic dimensionalities.
	lid_skew	Complexity	Skewness of local intrinsic dimensionalities.
	lid_std	Complexity	Standard deviation of local intrinsic dimensionalities.
	n_pcs	Complexity	Number of principal components that explain 90% of the variance.
	rv	Complexity	Relative variance of the dataset.
Graph configuration	graph_type	–	Graph type.
	IndexParams	–	Index parameter (*NN*).
	k_searching	–	Number of retrieved elements by a query.
	QueryTimeParams	–	Query time parameter (*R*).
Meta-targets	Recall	–	Average fraction of correct query answers retrieved.
	QueryTime	–	Average time for executing a query.
	DistComp	–	Average number of distance calculations during a query.

(*TMM-GS*), includes the same meta-instances than *GMM* plus meta-instances generated by the grid search performed over the input dataset. The third approach, the Tuned Meta-Model using Subsets (*TMM-S*), includes the same meta-instances than *GMM* plus meta-instances generated regarding a few subsets of the input dataset. With these fine-tuning procedures, we intend to improve the recommendations by ensuring that our meta-models know a dataset similar to the input one.

We present two meta-learning strategies for our framework in the following section. For both cases, we have generic and fine-tuned meta-models. We consistently achieved results by having an overall high accuracy. However, in the first strategy, our proposal generated some poor recommendations even with suitable predictive performances. On the other hand, the second strategy generates more accurate recommendations by employing additional meta-features and datasets and adopting a dataset-similarity-based meta-model generation, which produces a set of meta-models on a per dataset cluster basis.

4.5. Recommendation

As described in Section 4.2, the user receives recommendations by entering a new dataset and some requirements. The new dataset always is submitted for characterization. One of the user requirements is the method to use, which is one of the available generic or tuned strategies. If the method uses tuning, the dataset is submitted to performance measurement, and a tuned meta-model is induced using the meta-dataset, including the new meta-instances.

After that, the framework generates candidate recommendations, concatenating the user-provided k value, the new dataset's meta-features, and combinations of graph configurations. The framework selects the meta-models corresponding to the method and optimization goal (discussed later) indicated by the user and infers the candidates' meta-targets. The meta-targets associated with the corresponding graph configurations are forwarded to a filtering step.

The prediction filtering initially selects the candidates satisfying the user-requested minimum recall. For this task, we subtract the estimated error of the recall meta-model from the predicted recall. This operation is conservative regarding the prediction as we underestimate the recall based on the model error. Then, the candidates are ranked according to the user optimization goal, whose options are in Table 2. For query time and the number of distance computations, the candidates are sorted in increasing order of the corresponding meta-target. Ties are sorted by increasing value for *NN* to prioritize recommendations demanding less memory. For the memory usage optimization goal, the recommendation is based on the smallest indexing parameter (*NN*) value, as it represents the number of edges by vertex that a graph has. In this case, the candidates are sorted by *NN*, then by *R* as the number of restarts is directly proportional to both query time and the number of distance computations regarding the same graph type and *NN*.

Finally, the top recommendation is returned to the user. More than one recommendation is returned either in case of tied top recommendations or if the user asks for a larger number of

Table 2
Available optimization goals.

Optimization	Description
Memory usage	Lowest recommended *NN*
# of distance computations	Lowest predicted value
Query time	Lowest predicted value or smallest *R* value for same *NN* values

predictions to make her choice analyzing their expected performances. The user can consider more predictions helpful to analyze complex trade-offs. If it is desired to extend the recommendation to other index types, the candidate generation and prediction filtering should be adapted accordingly. The following section presents two recommender instantiations of the framework we developed.

5. Framework instantiations of meta-learning recommenders

In this section, we present two strategies to instantiate the meta-models in the proposed framework. We approach the problem of predicting different performance measurements as a multi-output regression problem. We developed a global and a dataset-similarity-based meta-learning instantiation. Section 5.1 introduces the global instantiation, which relies on meta-models regarding the whole meta-dataset. It was firstly presented in the previous version of this work [27]. Experimental results showed that the overall accuracy is good; however, the global instantiation produces some poor recommendations for specific cases. Therefore, we developed the dataset-similarity-based instantiation (Section 5.2), which significantly increases the meta-target prediction quality, yielding superior recommendations.

For both instantiations, we employed the same strategies for tuning.

5.1. A global meta-learning recommender instantiation

The global meta-learning strategy consists of fitting one regressor for each meta-target mt_i, as illustrated in Fig. 4. We consider it a global instantiation because we induce the meta-models from the entire meta-dataset. The input for the meta-learning phase is the meta-dataset, eventually enriched with tuning meta-instances. The meta-model generation task produces a model for each of the n meta-targets. In the inference phase, the *method* and *optimization goal* parameters provided by the user as a requirement indicate the meta-model to be selected (GMM, TMM-GS, or TMM-S) referring to the optimization goal. The meta-model regarding the recall is always selected because it is always used for filtering. The meta-model regarding the other performance meta-target is selected according to the optimization goal (e.g., query time or the number of distance computations). For instance, if the user wants to minimize query time, satisfying the minimum recall, the recommendation system will only use meta-models that predict recall and query time. The instantiation infers the meta-targets for the candidates previously generated and forwards the instances composed of the graph parameters and the corresponding meta-targets to the prediction filtering phase.

The global instantiation follows a standard procedure of meta-learning. The main innovations it brings are to apply meta-learning for recommending parameters for graph-based methods and the approaches for tuning. We refer to the generic and tuned meta-models of the global instantiation as *GMM*, *TMM-GS*, and *TMM-S*. In Oyamada et al. [27], we present experimental results for this instantiation, showing that it is simple but effective in many cases.

5.2. A dataset-similarity-based meta-learning recommender instantiation

The dataset-similarity-based strategy for instantiating the meta-models was designed to increase the predictive power of our recommendation framework. The idea consists of four major steps:

1. select the meta-features most relevant for a given meta-target;
2. create a dataset space according to these meta-features and apply a clustering algorithm;
3. induce meta-models tailored to the datasets from each cluster;
4. generate the recommendation for a new dataset using the meta-models from the cluster closest to the new dataset in the dataset space.

We illustrate the strategy in Fig. 5. Initially, we perform a feature selection step for each meta-target using the whole meta-dataset enhanced with data augmentation. The selected meta-features define a dataset space for each meta-target. After that, we apply a clustering algorithm and segment the meta-instances according to the cluster each dataset fall into, generating the clusters' meta-datasets. The strategy induces a meta-model for each meta-target and adds it to the meta-model base for each cluster.

This approach generates meta-models adjusted to the cluster's datasets. Therefore, these meta-models can provide highly precise predictions for new datasets with properties similar to the properties of the datasets in the cluster. Moreover, the tuning methods we propose for our framework have a higher impact as the fraction of tuned instances in the learning set is proportionally higher than the fraction in the global instantiation.

The inference step's inputs and outputs are the same as the previous section: the candidates and the parameterizations associated with the predicted meta-targets. The strategy finds the cluster closest to the new dataset and selects the meta-models from the cluster according to the *method* and *optimization goal* user parameters. Finally, the prediction generation outputs the candidate parameterizations associated with the predicted meta-target values to the framework's prediction filtering task.

We maintained the essence of the generic and tuned methods, keeping the fine-tuning approach the same. Thus, we named the dataset-similarity-based methods by adding a "plus" at the end of the name of the corresponding one, being *GMM+*, *TMM-GS+*, *TMM-S+*. The following subsections detail the main instantiation components.

5.2.1. Meta-feature selection

The meta-feature selection identifies the meta-features defining the similarity dataset space. After executing several analyses, we concluded that employing a large number of meta-features yields dataset spaces such that the dataset similarity poorly matches what we expected. As usual, feature selection can use alternative methods and cutting limits.

The proposed instantiation uses the feature importances provided by Random Forests to define the meta-features of the dataset space. Our choice is based on the robustness of the RF and experimental evaluations (see Section 7.3). We select the features whose importance is above a threshold. We identified that 0.9 is a suitable value for the threshold.

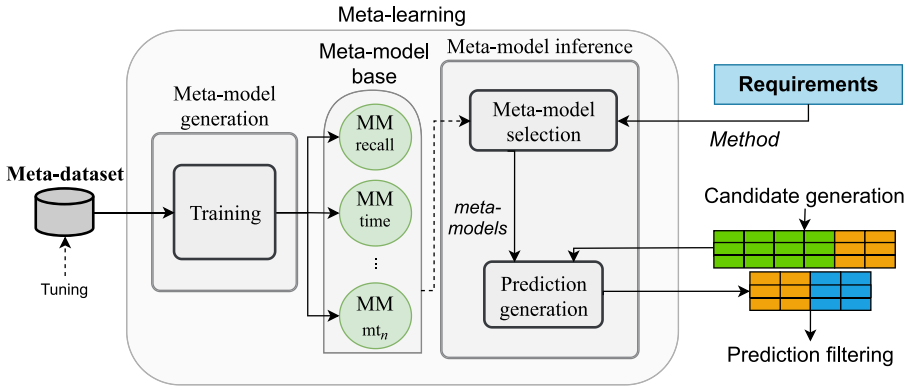


Fig. 4. The global instantiation of meta-models of the proposed framework.

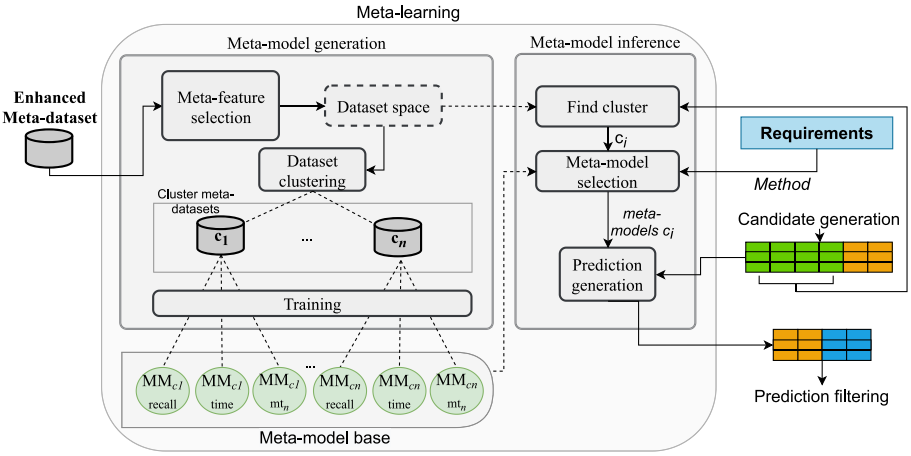


Fig. 5. The components and task flow of the proposed dataset-similarity-based recommender instantiation.

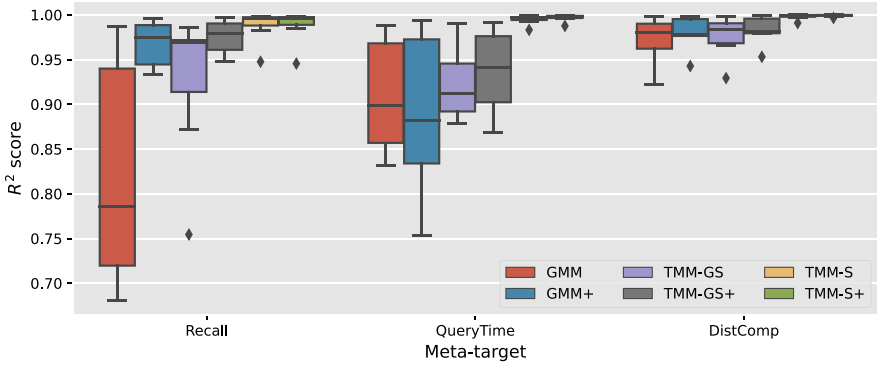


Fig. 6. Predictive performance distribution regarding each method and meta-target for all datasets.

5.2.2. Dataset clustering

We use the subset of relevant meta-features selected in the previous step and the Euclidean distance to represent the datasets in a metric space. We cluster similar datasets in this space employing a density-based clustering algorithm. Our option in this instantiation is the classical DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise) [71]. Although there are clustering alternatives, we opted for DBSCAN because clustering by density is better suited for our goal of training meta-models for similar datasets. Nevertheless, DBSCAN has the drawback of choosing suitable values for the parameters *eps* (the minimum distance to consider that two points should be connected) and *min_pts* (the minimum number of points within distance *eps* to define core points forming a cluster).

One could set the *eps* performing a grid search process, but this process demands a high computational cost. Instead, we propose defining the *eps* based on the pairwise distances between an element and its nearest neighbor. To do so, we build a k-NN graph, with $k = 1$, using a quick (approximated) construction method. The vertices are the datasets, and the edge weights the distance between the corresponding pair of vertices. We sort the edges by distance and pick the value defined by the 90th percentile to set the *eps*. Moreover, we set the parameter *min_pts*$= 2$, decreasing the chances of a dataset being an outlier. We estimate that using these parameter values, at least 90% of the datasets should belong to a cluster, ensuring a minimum variability in the meta-instances for most cases (i.e., meta-instances from two or more datasets). On the other hand, an outlier dataset (noise) forms a unitary

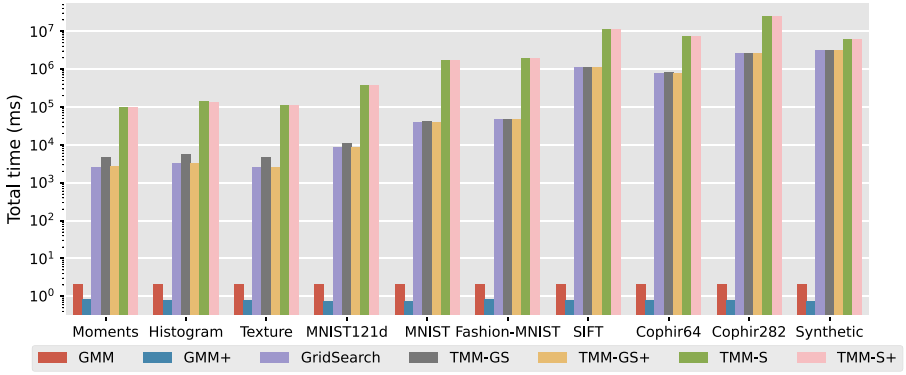
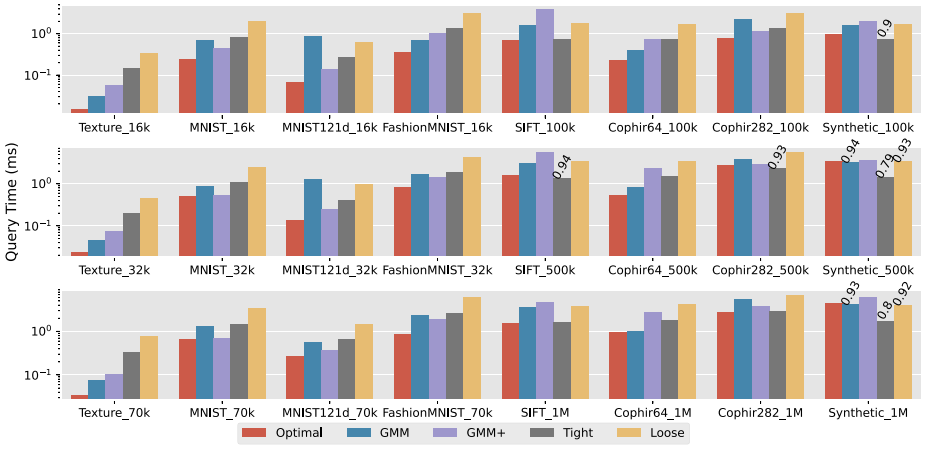
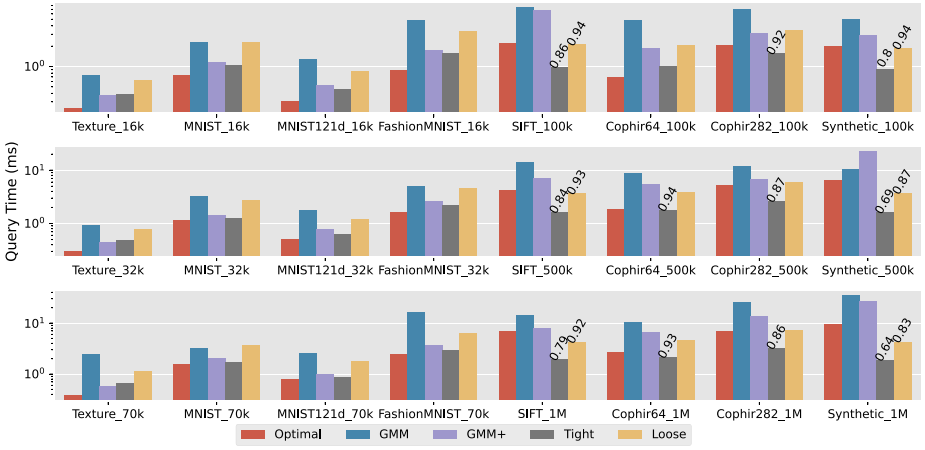


Fig. 7. The elapsed time for providing the final predictions for all meta-targets according to each method and dataset.



(a)



(b)

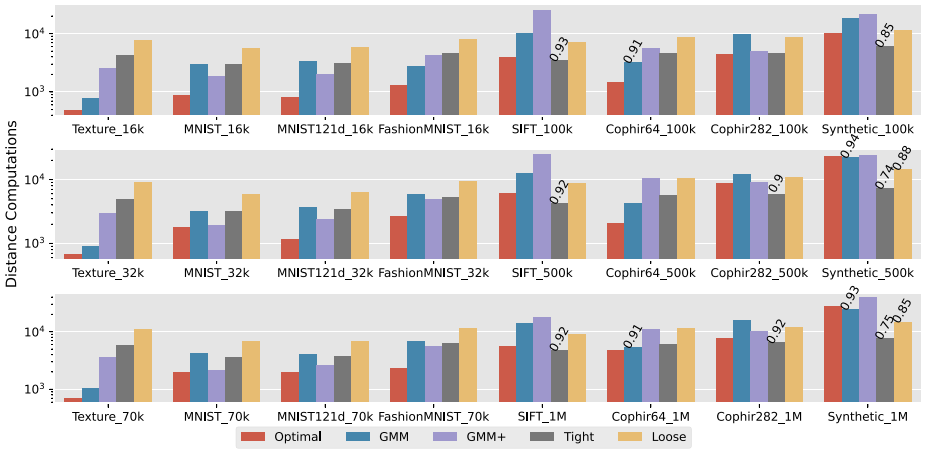
Fig. 8. Recommendations provided by generic methods optimizing the query time for (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

cluster as it may have better meta-models by not considering instances from datasets excessively dissimilar to it. This is the reason to pick the value in the 90th percentile.

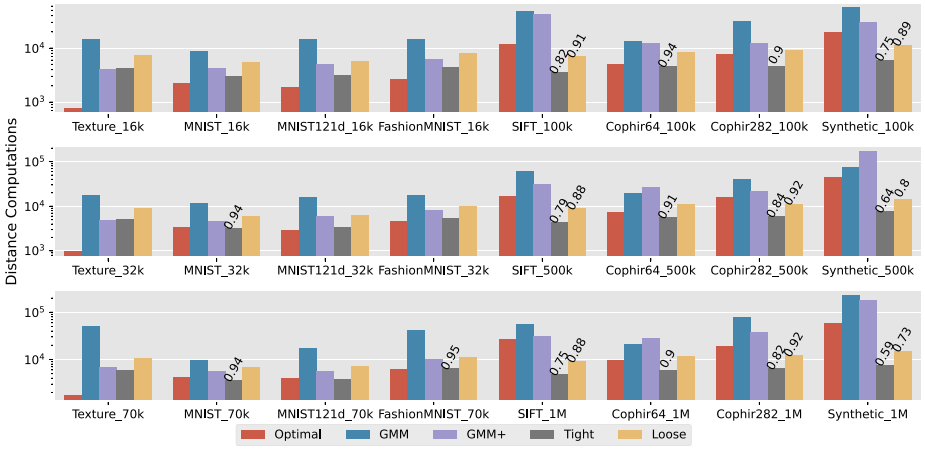
Subsequently, we employ the Random Forest as the base-learner for each cluster to induce a meta-model for each meta-target. The instantiation uses the default hyperparameters from Scikit-learn[3] and ensembles ten meta-models to average final predictions. We opted for RF due to its great prediction performances, fast training and inference, and simple parameterization, although alternative learning algorithms could be employed.

Employing cluster-based meta-models is also helpful to manage the evolution of the meta-dataset in the framework. Adding a few new datasets to the meta-dataset should have a localized impact on the framework. If the number of new datasets is reduced, there is a high probability they have little impact on the meta-feature importances. In this case, the dataset similarity space keeps the same, and only a few clusters should change. Therefore, it is only needed to rebuild the meta-models of the changed

[3] https://scikit-learn.org/.



(a)



(b)

Fig. 9. Recommendations provided by generic methods optimizing the number of distance computations for (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

clusters. On the other hand, if the number of new datasets is significant, the meta-model base should be completely refreshed, executing the whole process of dataset space definition, clustering, and meta-model training. Indeed, the framework's evolution should consider complete refreshes over time.

5.2.3. Meta-model inference

Given an input dataset, the first step to getting recommendations is finding the closest cluster to it. This is performed by employing a *k-NN* classifier. We found $k = 3$ to be a suitable value for our instantiation. Each of the k datasets retrieved votes for the input dataset cluster, and the majority defines the cluster. In the case of disagreeing votes (i.e., each dataset is from a distinct cluster), the selected cluster is the cluster of the dataset most similar to the input one.

In the case of tuning, the tuning meta-instances are added to the cluster's dataset to perform online training to generate the tuned meta-models. Finally, the instantiation selects the cluster meta-models, predicts all the performances (meta-targets) that the candidate graph configurations should achieve, and forwards them to the prediction filtering task.

6. Experimental evaluation

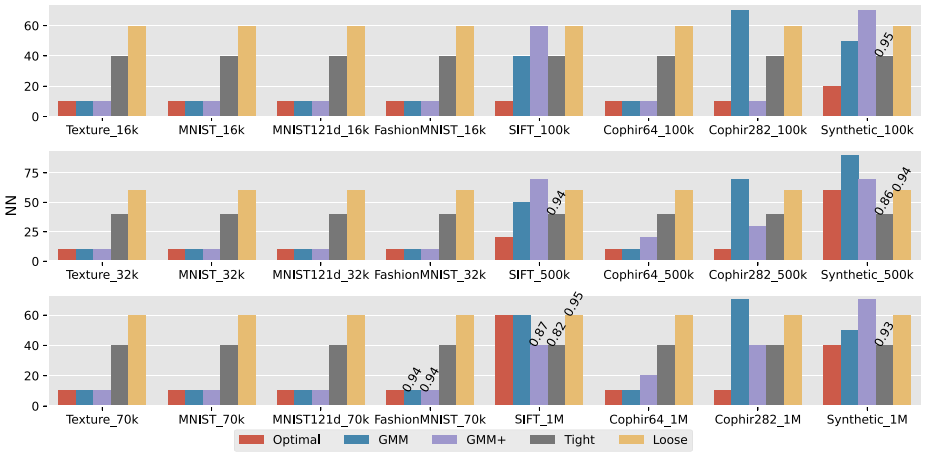
This section presents our experimental setup, the prediction performance of the proposed meta-models, and results comparing the recommendation effectiveness of the proposed methods with baselines.

6.1. Datasets and experimental setup

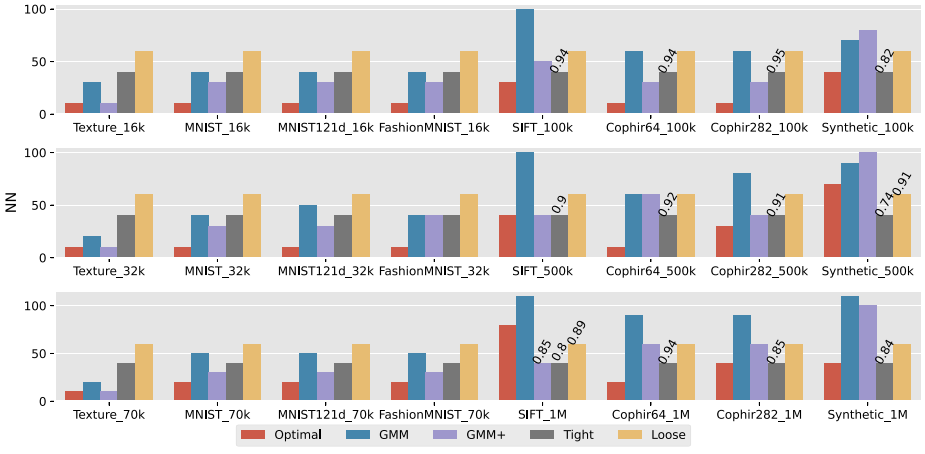
We have employed real and synthetic datasets to analyze the behavior of each graph-based method for different configurations. The real datasets contain features extracted from images, as follows: *Moments*, co-occurrence *Texture* and *Histogram*, which contain the corresponding feature vectors extracted from photos obtained from Corel, with dimensionalities 9, 16 and 32, respectively; *MNIST*, the pixels of a collection of images of handwritten digits comprising 784 dimensions; *MNIST121d*, which regards the original *MNIST* downscaled; *FashionMNIST*,[4] which is a more challenging version than the original *MNIST*; two feature combinations with different dimensionalities of a sample of one million elements from the CoPhIR dataset [72] (*Cophir64d* and *Cophir282d*), and *SIFT*, which is a collection of SIFT features (128 dimensions).

To increase the diversity of dataset characteristics, we generated synthetic data following a Gaussian distribution to manipulate different properties. Such properties are three values for size, dimensionality, number of clusters, and each cluster's distribution standard deviation, totaling $3^4 = 81$ synthetic datasets. The dataset generation was performed using the Python library Scikit-learn. We selected one of them to represent the synthetic datasets, called *Synthetic*, with 1,000,000 elements, 32 dimensions, and intrinsic dimensionality ≈25. The reason for choosing

[4] https://github.com/zalandoresearch/fashion-mnist.



(a)



(b)

Fig. 10. Recommendations provided by generic methods optimizing the memory usage for (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

this dataset is its high cardinality and intrinsic dimensionality close to the embedding dimensionality.

Table 3 summarizes all the values of the manipulated properties from synthetic datasets and an overview of the real datasets. We also partitioned the datasets to vary the cardinality. For the datasets up to 70,000 elements, we generated subsets starting from 500 elements and doubling the size up to the complete dataset: {500, 1k, 2k, 4k, . . . }. For the million-sized datasets, the subsets followed powers of ten, including half-sized datasets in between: {1k, 5k, 10k, 50k, 100k, 500k, 1M}. Thus, considering the complete datasets and their subsets, we totalize 375 datasets employed in this work.

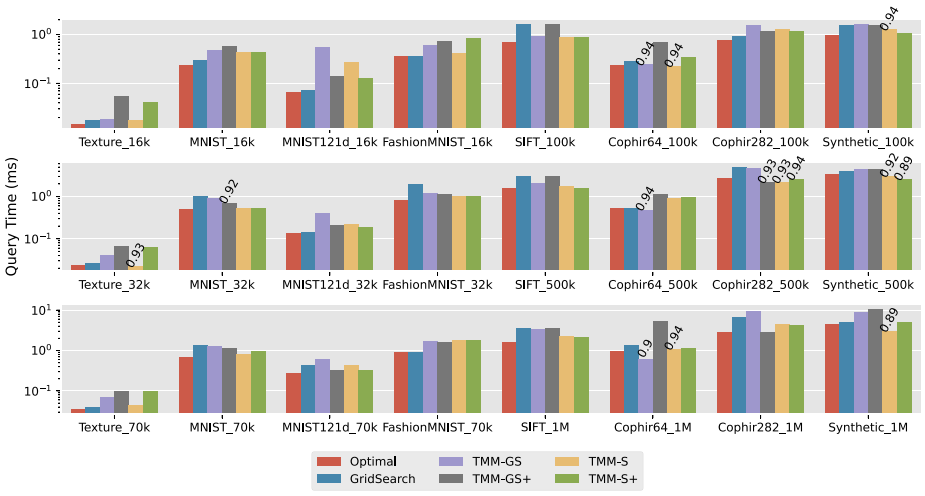
The graph-based methods selected in this work are the *BruteforceNNG*, *NN-Descent*, and *NSW*, using the *GNNS* search algorithm. We used implementations in the C++ library NMSLib (Non-Metric Space Library) [73]. The queries employed the Euclidean distance (L_2), and from each dataset, we removed 100 random objects to employ as *k-NN* query elements. The remaining ones were used to build the graphs. We used a superset of the results of the experiments carried on a previous work, which includes executions for combinations of the parameters $NN \in \{5, 10, 25, 40, 55, 70, 100, 130, 150\}$ and $R \in \{1, 5, 10, 20, 40, 80, 120, 160, 200, 240\}$. The *NN-Descent* and the *NSW* have specific construction parameters whose impact is not as crucial as the *NN* for query performances [22]. Thus, we fixed these parameters to values achieving a good general performance: $\rho = 0.5$ for the *NN-Descent*, and *efConstruction* = 100 for the *NSW*.

Table 3
The real and synthetic datasets employed in the work and their properties.

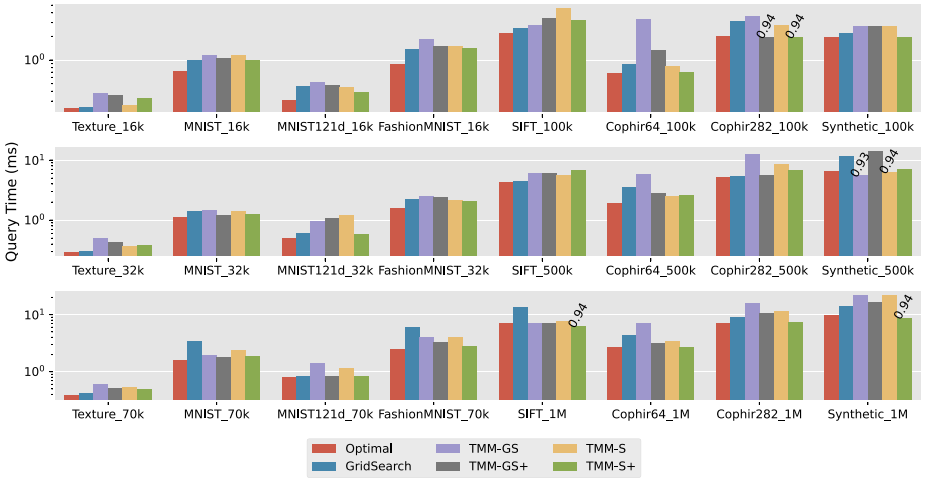
	Title	Size	Dimensions	Intrinsic Dim.
Real	*Moments*	68,040	9	6.39
	Texture	68,040	16	6.27
	Histogram	68,040	32	7.23
	MNIST121d	70,000	121	10.64
	MNIST	70,000	784	14.40
	FashionMNIST	70,000	784	16.05
	Cophir64d	1,000,000	64	13.59
	Cophir282d	1,000,000	282	20.33
	SIFT	1,000,000	128	21.46
	Synthetic	1,000,000	32	25.95
	Properties	Values		
Synthetic	Size	$\{10^4, 10^5, 10^6\}$		
	Dimensionality	{8, 32, 128}		
	Gaussian distribution	{1, 5, 10}		
	Number of clusters	{1, 10, 100}		

We used the Random Forest (RF) method from Scikit-learn to induce the meta-models, and a 5-fold Cross-Validation strategy to validate them. We employed the default hyperparameters[5] regarding the RF, where the main ones are *n_estimators* = 100, which refers to the number of decision trees in the model;

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.



(a)



(b)

Fig. 11. Recommendations provided by tuned methods optimizing the query time for (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

max_depth = *None*, which does not employ a maximum depth for the decision trees; and *criterion* = *squared_error*, which refers to the function to measure the quality for each split for in the decision trees. We also evaluated other methods, and the RF proved to be the best option for our case. We selected the RF for its excellent prediction performance, fast training and inference, simple parameterization, and capacity to evaluate feature importance.

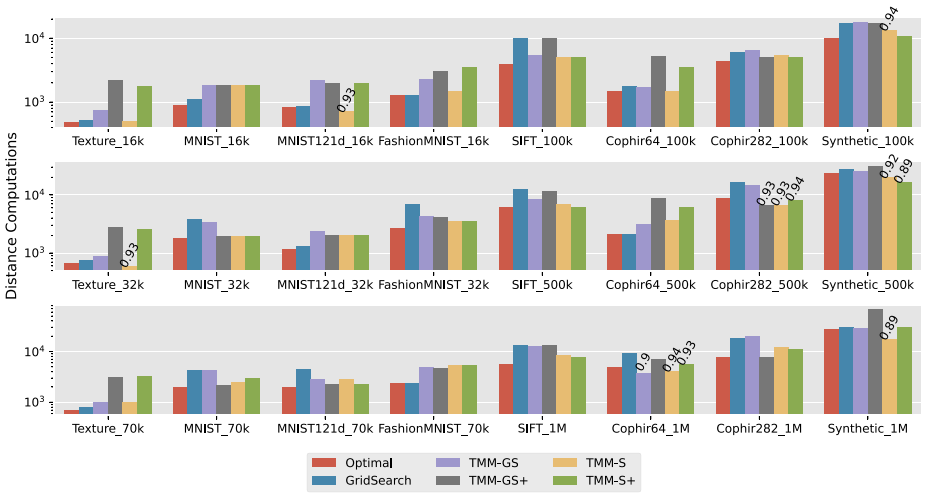
The experiments were carried out on an Intel Core i7 (32 GB RAM) with a single thread for all methods on an Ubuntu GNU/Linux 18.04.1 64 bits. The query time and distance computations were transformed into the log scale to smooth the values and improve the learning phase.

6.2. Analysis of the instantiations' prediction accuracy and execution time

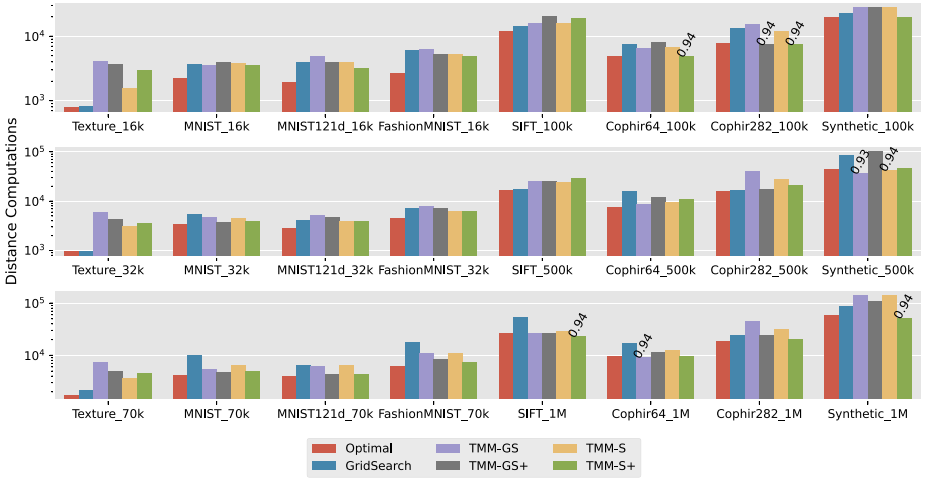
This section presents the results regarding generic and tuned methods for our recommendation framework following the global and the dataset-similarity-based instantiations. We provide experimental results about the prediction accuracy of the generated meta-models and compare the predictive performance and execution time regarding the methods following the two proposed instantiations. Regarding the execution time, we included the Grid Search as a baseline.

Recall that *GMM*, *TMM-GS* and *TMM-S* refer to the generic and tuned models (see Section 4.4) of the global instantiation and *GMM+*, *TMM-GS+* and *TMM-S+* are corresponding models of the dataset-similarity-based instantiation. The grid search procedure employed as baseline was the same as the one used to generate instances for tuning both *TMM-GS* and *TMM-GS+*, where a limited parameter space was defined with $NN = \{10, 25, 70, 150\}$ and $R = \{1, 10, 40, 120\}$. The parameter space for *TMM-S* and *TMM-S+* is the same as for *GMM* and *GMM+*, comprising more parameter combinations. However, for *TMM-S* and *TMM-S+*, the meta-targets were obtained by experiments on sub-datasets of decreasing sizes following an arithmetic or geometric progression depending on the original dataset size.

Fig. 6 shows the R^2 scores achieved by each method proposed in this work on our real datasets. We performed the entire learning process for each dataset to simulate a scenario where each of them would be an unknown dataset. We can notice that the dataset-similarity-based methods achieved significantly higher performance than the global methods, except for the *GMM+* for query time prediction. For the recall prediction, the dataset-similarity-based methods had the most significant improvement compared to the global methods. We see the recall as the principal meta-target to provide suitable recommendations because it defines the cutting point to attend to the user's requirements. Furthermore, the fine-tuning methods considerably



(a)



(b)

Fig. 12. Recommendations provided by tuned methods optimizing the number of distance computations for (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

improved the prediction accuracy for the three meta-targets. The methods tuned with grid search boosted the generic models' accuracy, and the methods tuned using subsets did it a lot more. The dataset-similarity-based *TMM-GS+* performed significantly better than the global *TMM-GS*. On the other hand, *TMM-S+* was only a bit superior to *TMM-S*, both achieving excellent accuracy.

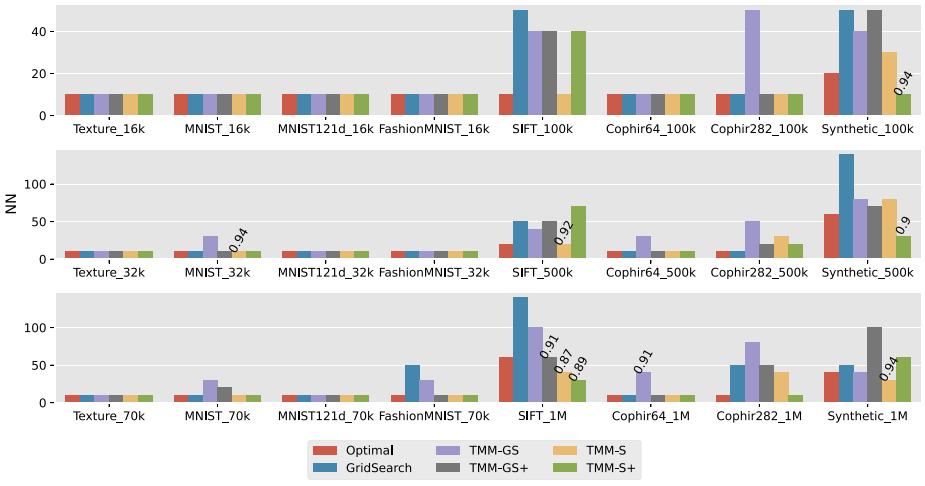
We measured the time needed to generate a recommendation to quantify the investment effort to get better recommendations. Fig. 7 shows the elapsed time for getting recommendations according to all methods evaluated in this work, including the Grid Search as a baseline. The Grid Search regards the time for running batches of experiments according to a limited parameter space. The *TMM-GS* and *TMM-GS+* methods are the sum of the Grid Search procedure and the elapsed time for training the meta-model and inferring recommendations. The same reasoning applies to *TMM-S* and *TMM-S+*, but employing more meta-instances from subsets of the input dataset.

As expected, *GMM* and *GMM+* are the fastest by far as they only count the inference time since the training phase is performed offline. We can notice *GMM+* is faster than *GMM* as it uses smaller meta-models. On the other hand, the tuned methods are two to six orders of magnitude more time-consuming than *GMM* and *GMM+*. Regarding the Grid Search and the tuned models using this strategy, the training time of *TMM-GS+* on top of the grid search time is negligible, while the overhead added by *TMM-GS* is significant. The overhead of *TMM-GS* comes from the size of the training meta-dataset, which is larger for *TMM-GS* than for *TMM-GS+*. As the training phase is performed over a few datasets belonging to a particular cluster in *TMM-GS+*, this time is significantly lower. Although the same reasoning also applies to TMM-S+, in the figure, *TMM-S* and *TMM-S+* appear to have the same execution time because the time of the performance measurement dominates the total time, making the difference regarding the training time irrelevant.

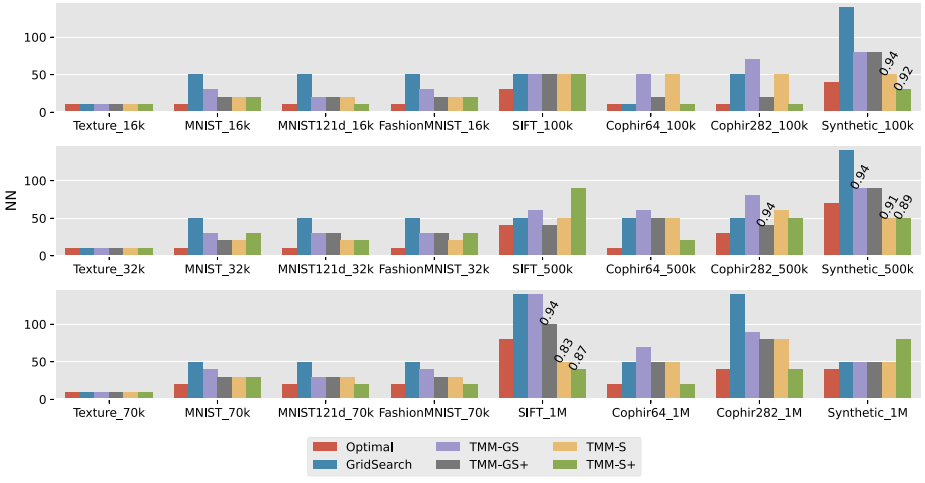
The tuning procedures are highly time-consuming. Therefore, the ideal scenario would be *GMM+* always providing the best recommendations as the time for recommendation is only the meta-model inference, which is fast. Nevertheless, it may be worth the tuning effort depending on the input dataset as the fine-tuning approaches improve predictions' quality. We present results regarding the final recommendations of each method for each real dataset in the next section.

6.3. Analysis of the recommendation effectiveness

This section discusses the effectiveness of the recommendation provided by our proposed methods. We divide this section into two parts: the first one for generic methods and the latter



(a)



(b)

Fig. 13. Recommendations provided by tuned methods optimizing the memory usage for (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

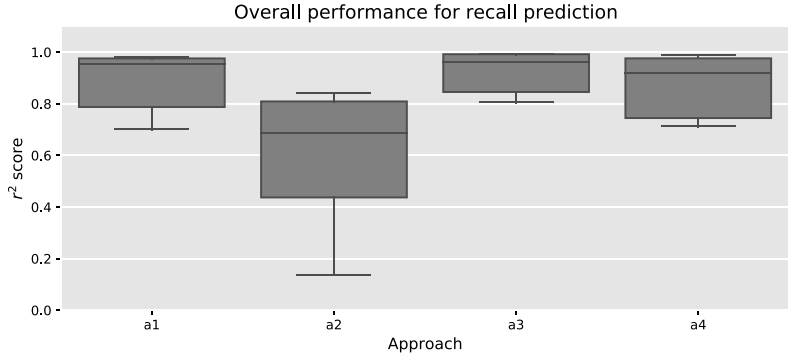


Fig. 14. Performance comparison using train-test procedures by alternating between the original and the augmented meta-datasets.

for tuned methods. Both parts include baselines and the optimal recommendation, which is the graph configuration that achieved the best performance, satisfying the minimum recall constraint. We do not show results for *Moments* and *Histogram* in this analysis because they performed similarly to *Texture* and due to their low dimensionality.

We considered the three optimization options (query time, memory usage, and the number of distance computations) and present the results regarding each of these metrics by considering the number of retrieved objects $k = \{1, 30\}$, different required recall values: 90%, 95%, and 99%, and different cardinalities for each dataset. However, in this section, we are showing only recommendations regarding the required recall of 95%, whereas the remaining ones are attached as complementary material in Appendices A and B. Notice that, in the following figures, when a method fails to satisfy the recall constraint, we show the actual recall for the recommendation on the corresponding bar to indicate how far it is from the required value. These cases represent wrong parameter recommendations because the predicted recall is above the requirement, but its true value is not.

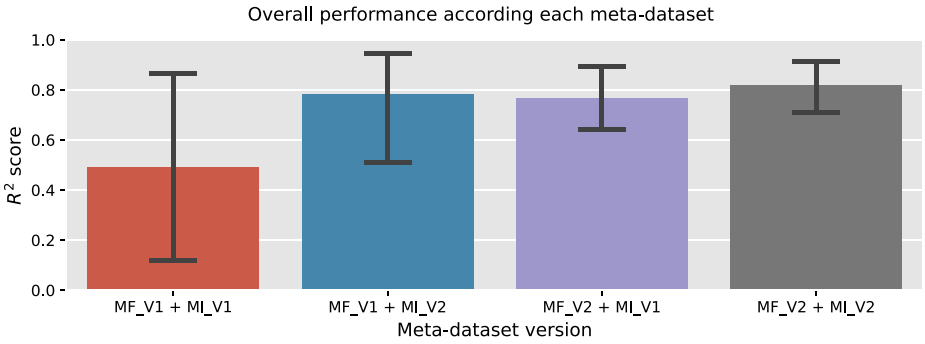


Fig. 15. Enhancement promoted by adding new meta-features and/or meta-instances from new datasets.

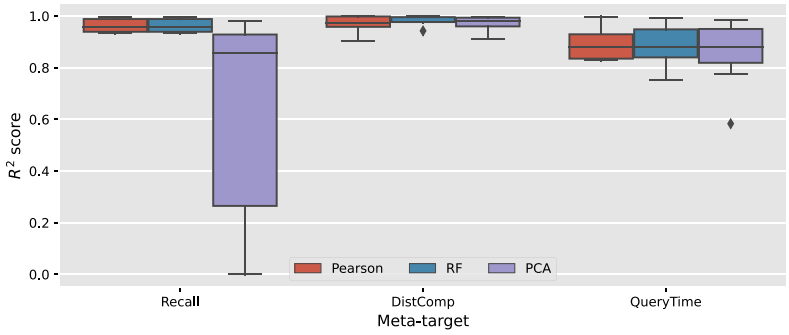


Fig. 16. Feature selection methods considering the *eps* selected according to our proposal.

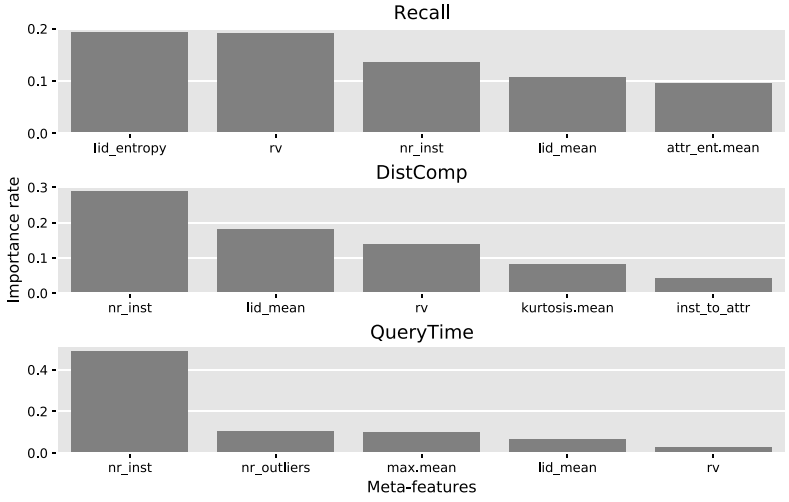


Fig. 17. The meta-features defining each meta-target's dataset space according to the RF method and their importances.

6.3.1. Generic methods

This section presents the generic methods' effectiveness, comparing the *GMM* and *GMM+* methods with the optimal results and two baselines. The baselines *Tight* and *Loose* are defined by averaging the parameter values regarding the top-10% configurations for all datasets, according to the given optimization metric. First, we filter the results satisfying the desired recall, sort them by the optimization metric, and select the top-10% configurations. Subsequently, we identify the most frequent graph type into that subset to define it as the recommended one. The subset of the top-10% recommendations corresponding to the selected graph type is used to define the two baselines, as follows.

- *Tight*: in this recommendation, we set the recommended *NN* by averaging the *NN* in the subset; the recommended *R* is the average value for the entries with the established *NN*. We call this recommendation *tight* because it aims at choosing values that are the best on average but underestimated in some cases.
- *Loose*: this recommendation takes the largest value for the *NN*; the recommended *R* is also the average value for the entries with the established *NN*. This recommendation is considered *loose* as it takes a conservative approach, getting the "safest case" in the subset of top recommendations.

Fig. 8 shows recommendations optimizing the query time, where (a) regards recommendations for $k = 1$ and (b) $k = 30$ as the numbers of retrieved elements. Overall, both methods *GMM* and *GMM+* perform greatly and always provide correct recommendations, except *GMM* for *Synthetic* 500*k* and 1*M* with $k = 1$. However, *GMM+* is usually closer to the *Optimal*, which highlights the improvements achieved by the new proposed method.

For *Texture*, *FashionMNIST*, and *SIFT* with $k = 1$ and different cardinalities, the *GMM+* is slightly outperformed by the *GMM*, mainly for the first two ones. The baselines are competitive in a few cases, specifically for datasets *MNIST*, *MNIST* 121, and *FashionMNIST* regarding recommendations with $k = 30$. Nevertheless, the *GMM+* performs better mainly for datasets whose cardinality is 1*M*. Assuming that the hardness of a dataset can be estimated by its dimensionality and intrinsic dimensionality, we notice *GMM* providing recommendations farther from the *Optimal* for complete datasets like *FashionMNIST*, *SIFT*, *Cophir282d*, and *Synthetic*. This fact highlights the enhancement of the new proposal *GMM+*, which is able to recommend settings closer to the *Optimal*.

Following the same pattern, Fig. 9 shows recommendations optimizing the number of distance computations. Again, the *GMM+* outstands for not providing any wrong recommendation. However, for *Texture* regarding all cardinalities and k values, its recommendations are distant from the *Optimal*. The same occurs for *SIFT* 100*k* with $k = 30$ and *SIFT* $= \{100k, 500k\}$ for $k = 1$. Similarly, the baselines provide several wrong recommendations for datasets with cardinality greater than 100*k*.

Lastly, Fig. 10 shows recommendations optimizing the memory usage. This time, *GMM+* provided a few wrong recommendations, more specifically for *FashionMNIST* 70*k* with $k = 1$ and *SIFT* 1*M* for both k values. However, the optimal recommendations were frequently achieved for this optimization metric. Furthermore, in general, the *GMM+* outperforms all the other methods in most cases. There are only a few recommendations where the *GMM+* is outperformed by the *GMM* or the *Loose* baseline. Nevertheless, the difference among the recommended settings in such situations is low.

6.3.2. Tuned methods

This section presents the results achieved by the methods using tuning: *TMM-GS*, *TMM-GS+*, *TMM-S*, and *TMM-S+*. We compare them to the optimal case and employ the classic Grid Search as a baseline.

Fig. 11 shows recommendations optimizing query time. Overall, the tuned methods were able to overcome the Grid Search or perform equally. In most cases, the *TMM-S+* provides recommendations quite close to the *Optimal*, although it also provides some wrong recommendations. However, it is remarkable that in such cases the true recall achieved by the recommended setting has a difference less than 0.01 from the required recall. Regarding the methods tuned with instances generated from a grid search procedure, *TMM-GS* and *TMM-GS+*, the latter one performs slightly better in most cases and much better for both k values and all cardinalities of *Cophir64d*. On the other hand, the methods tuned with subsets of each the input dataset perform similarly but the *TMM-S+* is slightly better in most cases.

For recommendations optimizing the number of distance computations, as illustrated in Fig. 12, all methods perform poorly for both k values and all cardinalities regarding the *Texture* dataset. For $k = 30$, the *TMM-S+* in most cases performs excellently, always providing recommendations close to the *Optimal* or at least overcoming the Grid Search, and presents a negligible error rate regarding wrong recommendations. On the other hand, for $k = 1$, it present higher error rates and it is eventually overcomed in some cases, for instance for *Cophir282d* 1*M* and *FashionMNIST* 70*k*. Nevertheless, this method can be considered the best one mainly for datasets with cardinality of 1*M*.

Regarding the recommendations optimizing memory usage, all methods recommended optimal settings for most datasets. The only exceptions, which could not receive optimal recommendations, are the *Synthetic* and *SIFT* regardless of the k value and cardinality. The outstanding method again is the *TMM-S+*, which often achieved the *Optimal* and outperformed the Grid Search. There were several wrong recommendations for *SIFT* 1*M* with both k values. Considering the most expensive datasets, i.e. higher cardinalities and dimensionalities, for finding optimal settings, the proposed methods outperform the Grid Search (e.g. *TMM-S+* for *Cophir282d*) or perform likely (e.g. *Synthetic* with $k = 30$) (see Fig. 13).

Table 4
Summary of approaches regarding the proposed data augmentation strategies.

Approach	Training dataset	Testing dataset
a1	Original	Original
a2	Original	Augmented
a3	Augmented	Original
a4	Augmented	Augmented

6.4. Summary of the results

This section presented experiments comparing the global and tuned methods to the optimal case and baselines. The results showed we improved the predictive performance, allowing our system to provide more robust recommendations. A remarkable achievement here was the high-quality recommendations of the *GMM+*, being the fastest method proposed in this work. It outperformed fixed general recommendations in most cases. We also highlight that our tuned methods *TMM-GS+* and *TMM-S+* provided the best recommendations, in two levels of increasing accuracy and cost. These methods achieved the optimal many times and frequently outperformed the *Grid Search* regarding all optimization metrics. Even in cases where our methods produced wrong recommendations, the difference between the required recall and the actual value achieved by the recommendation was quite low, in general. The results corroborate that this work proposed a framework and instantiation strategies that advance the state-of-the-art regarding selecting proximity graphs and their parameters.

7. Analysis of the supporting techniques

In this section, we discuss how the techniques supporting the recommender instantiations behave and impact the performance of the approach. We show how we used the data augmentation technique to enhance the effectiveness of our proposal. We also present results supporting the options we chose to build the dataset space, including alternatives for selecting relevant meta-features according to a specific meta-target as well as cluster information and the predictive power for each alternative.

7.1. Impact of the data augmentation technique

This section shows the effectiveness of proposed data augmentation to the meta-dataset enhancement. We employed the *GMM* method (the generic meta-model of the global instantiation) to assess the overall predictive performances by varying the training and testing sets using data augmentation. We considered four different approaches, summarized in Table 4. The goal of this study is to understand the impact of augmenting or not each set (training or/and testing). In the table, *original* refers to only using the original meta-instances, whose meta-targets were extracted from graph-based methods by running batches of experiments. Conversely, *augmented* refers to meta-instances weak-labeled using interpolated meta-target values.

Fig. 14 shows the R^2 scores regarding the real datasets employed in this work. Comparing *a1* and *a2*, we can notice that the regressors trained using only the original meta-instances

failed to predict many interpolated values. This issue limits the proposal of recommending parameter values unseen by the meta-models during learning. We acknowledge that the results presented herein have the bias of using interpolated values in the testing phase, which means the ground truth is estimated. However, we still consider the results valid because we achieved negligible errors while evaluating the interpolated results using truth values, as mentioned in Section 4.3.3. Comparing *a1* with *a3* and *a2* with a4, it is clear that the data augmentation improved the meta-models' performance as the testing set of each pair of approaches is the same. Moreover, analyzing *a4*, we can check that the performance drop from *a3* to *a4* is significantly smaller than the performance drop from *a1* to *a2*. These results show that the proposed augmentation technique makes meta-models more robust to predict the meta-targets for instances employing parameterizations not considered during the framework's performance measurement step, including the alternative tuning path. Therefore, we consider that our augmented meta-dataset is valid for performing recommendations in a vast parameter space not limited to samples actually measured on the graph-based methods.

7.2. Impact of the additional meta-features and datasets

It is expected that the prediction accuracy of the meta-models gradually increase with additional meta-features and datasets. This section shows a test quantifying how much these two aspects improve the accuracy.

We simulate the evolution of the meta-features by adding more features of the complexity category (see Table 1). Specifically, we consider two sets of meta-features:

- *MF_V1*: all features from the categories statistical, general and information theory plus the intrinsic dimensionality (*lid_mean*);
- *MF_V2*: all features in *MF_V1* plus the remaining ones from the complexity category.

Likewise, we simulate the increase of the dataset base using two sets of datasets from Table 3:

- *MI_V1*: all datasets except *FashionMNIST* and *MNIST121d*;
- *MI_V2*: all datasets.

We analyzed how the meta-dataset enhanced for every combination: *MF_V1* + *MI_V1*,[6] *MF_V1* + *MI_V2*, *MF_V2* + *MI_V1*, and *MF_V2* + *MI_V2*. In this analysis, we also employed the *GMM*, measuring the R^2 score achieved after training the meta-models using the following meta-dataset versions.

Fig. 15 shows the enhancement achieved by the GMM regarding its R^2 performances. We can notice a meaningful improvement in each version. Either for only adding meta-features or for only adding meta-instances from new datasets, the improvement in the R^2 score is evident. Nonetheless, we achieved the highest average performance with the lowest variation by employing additional meta-features and datasets. These results confirm that the meta-knowledge evolution contributes to higher average predictive performances, leading to improved recommendations. Therefore, we consider the enhanced meta-dataset using the complete meta-feature set and all datasets.

[6] This was the set of meta-features employed in our previous work [27].

7.3. Analysis of feature selection techniques to generate the dataset-similarity-based meta-models

This section discusses the effect of different feature selection (FS) methods on defining the similarity space for clustering the datasets. We evaluated three different techniques on the complete meta-dataset to perform feature selection: the feature importances provided by Random Forests, the Pearson correlation, and the Principal Component Analysis (PCA). We perform cross-validation on the meta-dataset for the RF technique and average the returned importances by each meta-target's model. For the Pearson correlation, we average the correlations between the meta-feature and each meta-target for every meta-feature. We consider the absolute value calculated for each meta-feature for these two approaches and select all the meta-features that lie above the 90th percentile. Lastly, for PCA, we sort the explained variances of the principal components and select the first ones needed to represent 90% of data variability.

After that, we performed the clustering process as follows. As we mentioned in Section 5.2.2, we defined the DBSCAN's *eps* value by evaluating the distance distribution among all datasets using a *k-NN* graph with $k = 1$ and picked the value corresponding to the 90th percentile. Although our proposal performs a single clustering per meta-target, the clustering procedure was performed for each real dataset and each meta-target in this analysis. That is, given n datasets, the clustering was made considering $n-1$ datasets, and the remaining one was treated as the input dataset. This approach was adopted because the clusters may vary according to each entry, and we wanted to grasp the magnitude of the changes.

As we performed this clustering procedure in a leave-one-out fashion, we aggregate the obtained information about it in Table 5. The first two columns show the average number of clusters and the average percentage of outliers. The remaining ones refer to the number of datasets per cluster, averaging the mean, standard deviation, maximum, and minimum statistical metrics. Notice that there is a single line for PCA in the table because it is an unsupervised method and all meta-targets use the same meta-features. For instance, from the 375 datasets employed in this work and using PCA, we have an average of 70 clusters, 17% of the datasets classified as outliers, and a mean of 3.7 datasets per cluster. The cluster information reveals that the dataset space is diverse in terms of cluster size and the number of clusters for different meta-targets. This result supports our idea that employing clustering adapts better to the dataset variability than always selecting a fixed number of similar datasets.

Finally, we evaluated the performance of the cluster-based meta-models, taking the real datasets used in this work as the input datasets. For every real dataset as input, we identified the cluster closest to it, induced a meta-model per meta-target using the remainder datasets in the cluster, and evaluated the meta-models' predictive power using the instances of the input dataset. We used all real datasets to perform this analysis, and the R^2 scores per feature selection technique (averaged for RF and Pearson) are shown in Fig. 16. For recall, the meta-models using Pearson and RF achieved almost the same performance because most of the selected meta-features were the same while the results were wretched using PCA. The meta-models generated for the remaining targets using the RF feature selection presented superior performance than the other methods. Therefore, we adopted the RF-based feature selection in the dataset-similarity-based recommender instantiation.

It is worth highlighting that, in general, the cluster-based meta-models are significantly more accurate than the corresponding global meta-models. We can check this fact comparing the R^2 score box-plot regarding RF for recall in Fig. 16 and the

Table 5
Cluster information on dataset spaces defined by the feature selection methods.

FS method	Meta-target	n clusters	Outliers (%)	avg. mean	avg. std	avg. max	avg. min
PCA	All	70.5714	17.25	3.6962	1.3792	9.0000	2.0
Pearson	DistComp	47.0000	17.21	5.5537	4.7034	18.0000	2.0
	QueryTime	44.2857	17.76	5.8551	7.2423	36.2857	2.0
	Recall	61.0000	17.25	4.2765	2.7496	12.0000	2.0
RF	DistComp	65.1429	0.2454	3.6272	2.5817	12.0000	2.0
	QueryTime	81.0000	31.45	2.6297	1.1984	8.0000	2.0
	Recall	61.0000	17.25	4.2765	2.7496	12.0000	2.0

box-plot of the approach *a4* in Fig. 14. These two boxplots refer to the average performance of the model(s) regarding recall for the same datasets. The boxplots show the superiority of the cluster-based meta-models compared to the global meta-model for this meta-target.

Fig. 17 shows the selected meta-features for each meta-target according to the RF method. Coincidentally, the number of meta-features was five for all meta-targets. In summary, the meta-features *nr_inst*, *lid_mean*, *rv* were considered the most important ones for all meta-targets. For recall, the two most significant meta-features refer to complexity metrics employed in this work, which is meaningful as the recall rate is usually inversely proportional to the dataset complexity. On the other hand, for distance computations and query time, the most important meta-feature was the number of instances. This is also meaningful since it is well known that these metrics are directly related to the dataset size.

8. Conclusion

This article presented a novel intelligent tuning approach for graph-based methods for similarity searching. We propose a meta-learning-based framework to recommend the best graph-based method and its optimal construction and searching parameters for a new complex dataset, satisfying specific requirements provided by the user. The contribution includes two instantiation strategies of our framework: a global and a dataset-similarity-based.

In this work, we extended our previous contribution [27] by enhancing the meta-knowledge gathering process and by proposing the dataset-similarity-based strategy. The meta-knowledge enhancement (i) employs new meta-features that present more relevance for the approximate search problem, (ii) includes more datasets to increase the data variety, and (iii) introduces a data augmentation technique to the meta-dataset. The dataset-similarity-based strategy performs a clustering procedure to instantiate meta-models for each cluster, adjusting the recommendation relying on common properties among datasets.

We separate our instantiation strategies into generic and tuned methods. The generic methods do not require extra time for the recommendation. In general, the proposed generic methods *GMM* and *GMM+* performed better than two baselines simulating an expert user, frequently reaching values close to the optimal for all optimization metrics. The tuned methods need an investment of a certain amount of time to acquire more precise recommendations. We introduced two tuning variations with increasing accuracy and cost: the Tuned Meta-Model using Grid Search methods (*TMM-GS* and *TMM-GS+*), and the Tuned Meta-Model using Subsets methods (*TMM-S* and *TMM-S+*). In general, our tuned methods provided precise recommendations, outperforming the *Grid Search* approach in most cases. When the meta-models provided underestimated predictions, the difference between the recommended and the required recalls was slight, showing the overall suitability of our approaches. Among the proposed methods, the dataset-similarity-based instantiations were superior to the global ones.

In future work, we intend to address the following ideas. We plan to adapt our proposed framework to other index structures (e.g. tree-based or hash-based methods) and evaluate its performance in other metric spaces beyond the euclidean. To improve the dataset representation, we also want to explore more metrics of the "hardness" of datasets regarding the nearest neighbor problem. Finally, we intend to explore existing techniques in the machine learning community regarding model generalization, for example, zero-shot learning.

Declaration of competing interest

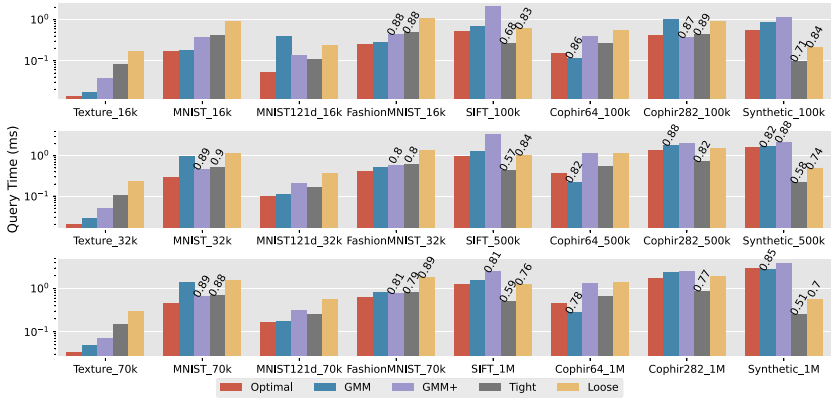
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

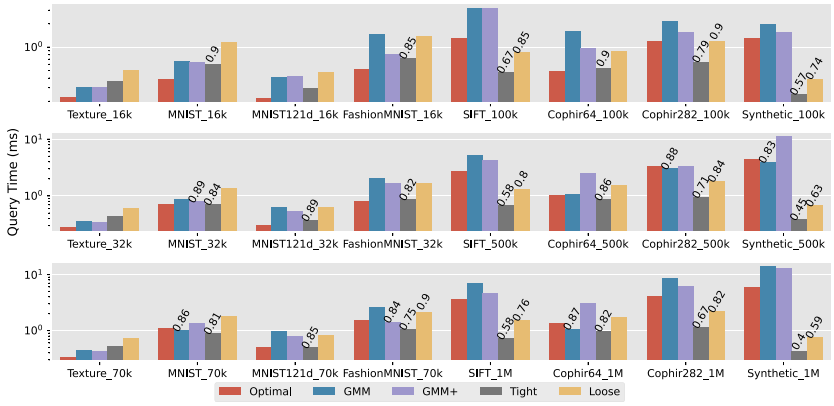
This work has been supported by the Brazilian funding agencies CAPES (Coordination for the Improvement of Higher Education Personnel), CNPq (National Council for Scientific and Technological Development), and also has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825041.

Appendix A. Recommendation effectiveness of generic methods

This section presents recommendations provided by generic methods regarding other required recall values. We do not include recommendations optimizing distance computations because the overall behavior was similar to recommendations optimizing query time.



(a)



(b)

Fig. A.18. Recommendations provided by generic methods with a required recall of 0.90 optimizing the query time regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

A.1. Required recall 0.90

Optimizing query time in Fig. A.18, we can see higher error rates, for instance with $k = 1$ for *FashionMNIST* and *SIFT*, and the *GMM* providing wrong recommendations more frequently. However, in general, *GMM+* still performs well by providing recommendations closer to the optimal for easier datasets, like *Texture*, and not making wrong recommendations for harder datasets, like *Synthetic*.

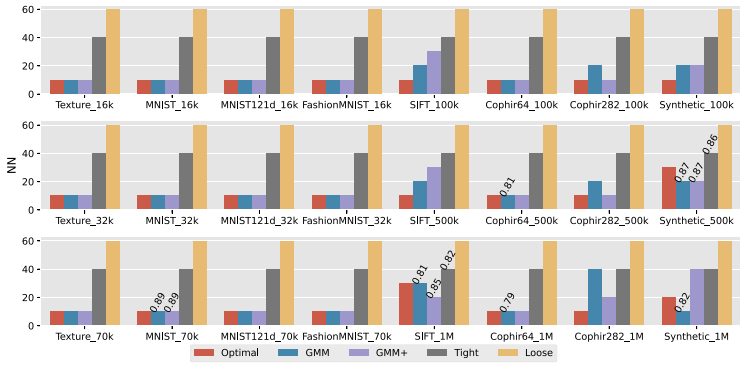
For memory usage in Fig. A.19, our methods present an outstanding performance in overall, frequently providing optimal recommendations. However, there are wrong recommendations with high error rates for *SIFT* and *Synthetic*, highlighting the hardness of such datasets according to their intrinsic dimensionalities. The baselines can provide correct recommendations for most dataset but always farther away from the optimal.

A.2. Required recall 0.99

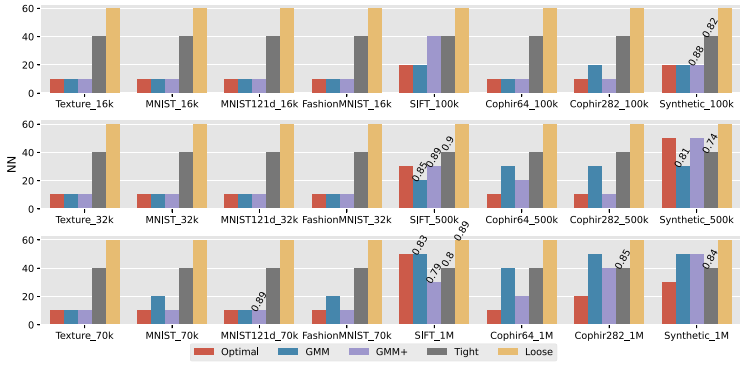
The required recall of 0.99 is the most difficult one provide recommendations because the model must be able to perform predictions with an error of 0.01 at most. Optimizing query time in Fig. B.24 we can notice the models frequently providing wrong recommendations. However, we can see the *GMM+* presenting a remarkable performance for harder datasets with $k = 1$. Considering $k = 30$ the models provided several wrong recommendations but close to the required recall value.

Optimizing memory usage in Fig. A.21, we can see *GMM+* frequently providing wrong recommendations, but with low error rates in general. However, for $k = 1$ its performance is remarkable mainly for harder datasets, since most of its recommendations are correct in this specific case. The baselines are not competitive in general for both optimization metrics.

It is remarkable that our quick methods, mainly the *GMM+*, can provide good recommendations even considering those that do not satisfy the required recall due to the low error rate. In a real use case, the needed time to acquire such recommended settings for a dataset is order of magnitudes less than the classic grid search procedure, as we illustrated in Fig. 7 (see Fig. A.20).

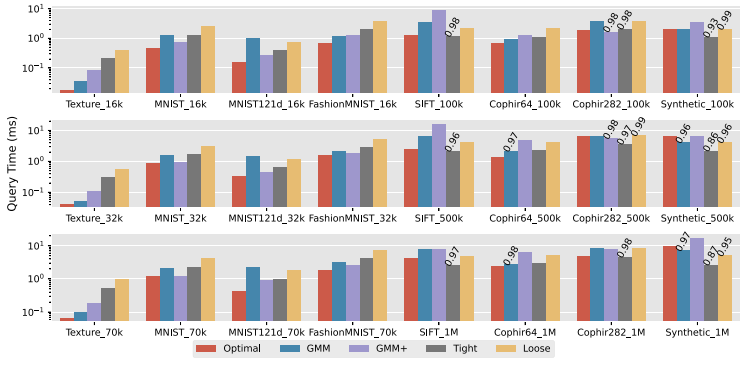


(a)

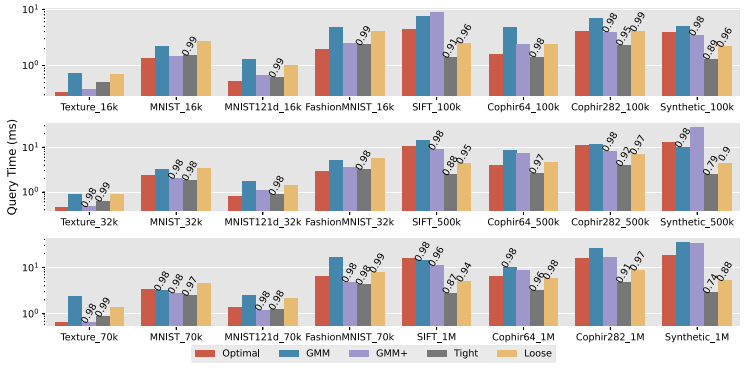


(b)

Fig. A.19. Recommendations provided by generic methods with a required recall of 0.90 optimizing the memory usage regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

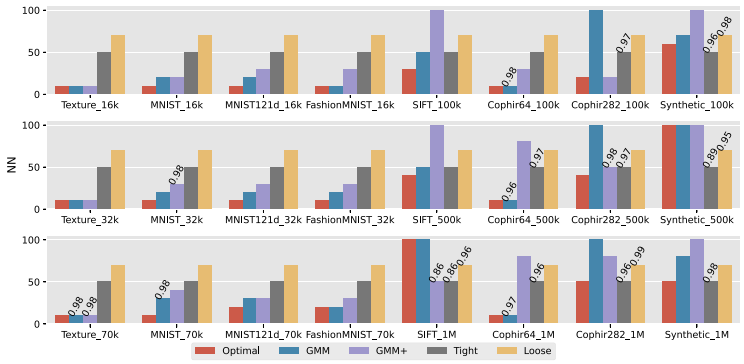


(a)

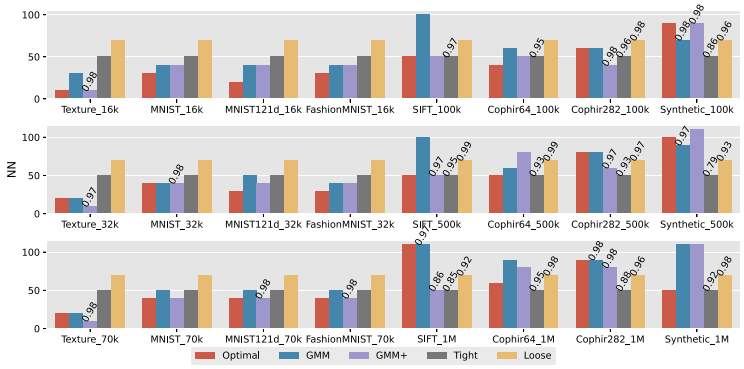


(b)

Fig. A.20. Recommendations provided by generic methods with a required recall of 0.99 optimizing the query time regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

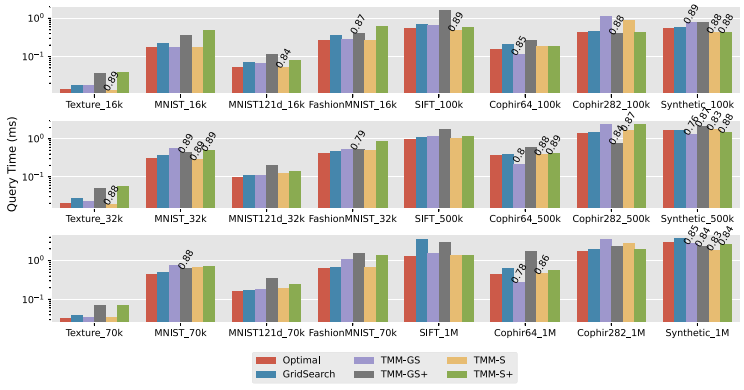


(a)

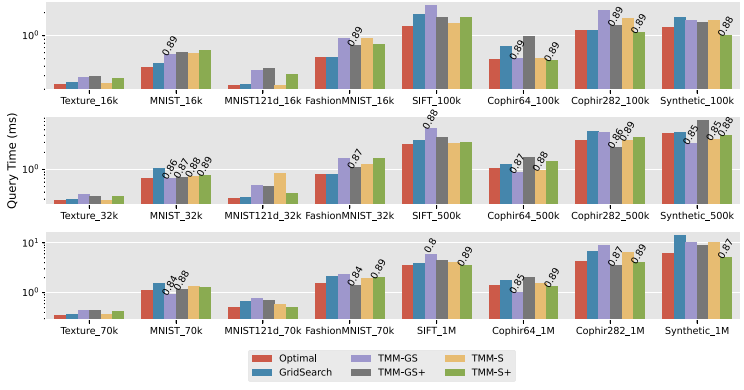


(b)

Fig. A.21. Recommendations provided by generic methods with a required recall of 0.99 optimizing the memory usage regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

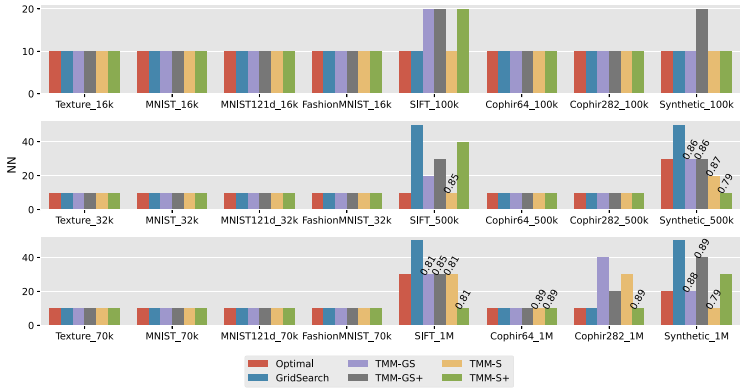


(a)

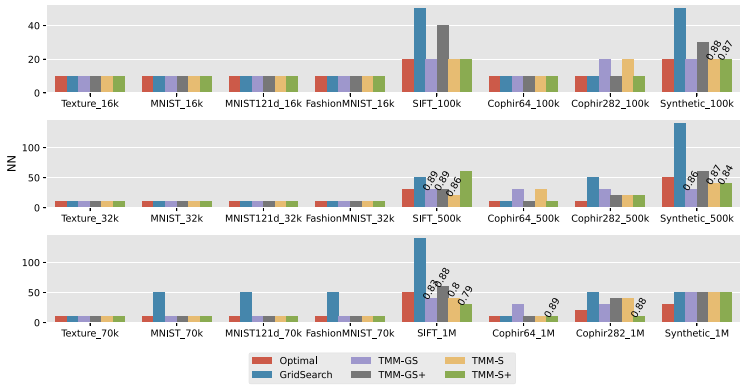


(b)

Fig. B.22. Recommendations provided by tuned methods with a required recall of 0.90 optimizing the query time regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

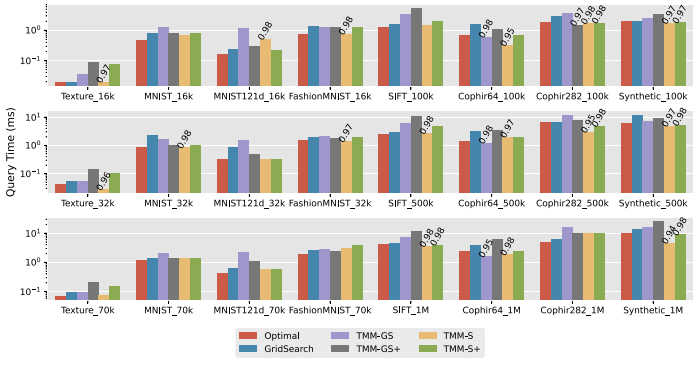


(a)

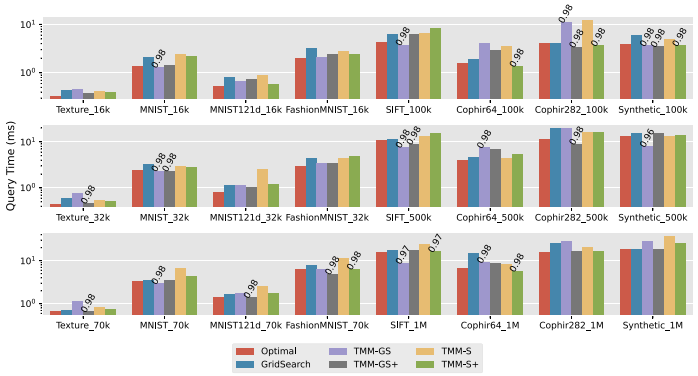


(b)

Fig. B.23. Recommendations provided by tuned methods with a required recall of 0.90 optimizing the memory usage regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

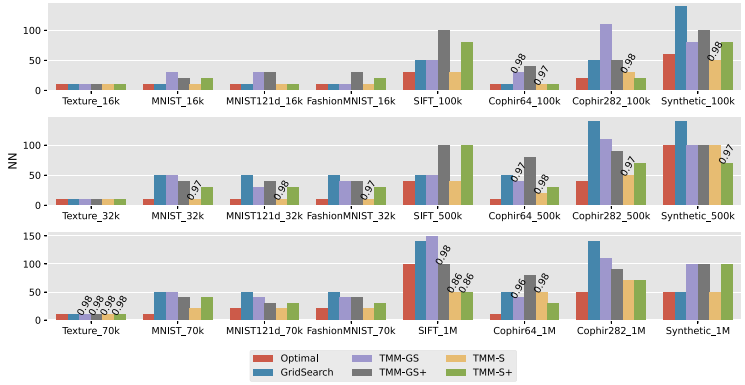


(a)

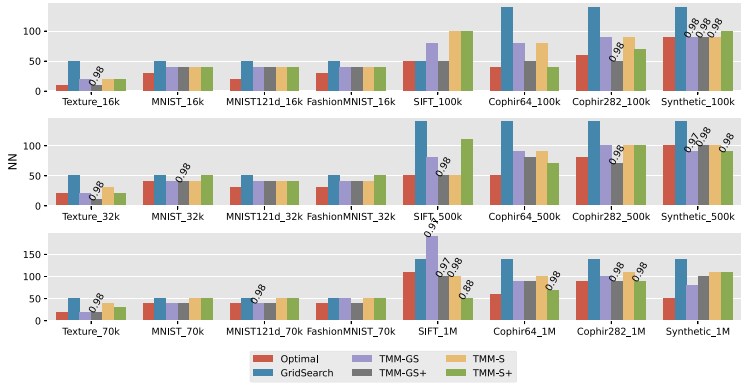


(b)

Fig. B.24. Recommendations provided by tuned methods with a required recall of 0.99 optimizing the query time regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.



(a)



(b)

Fig. B.25. Recommendations provided by tuned methods with a required recall of 0.99 optimizing the memory usage regarding (a) $k = 1$ and (b) $k = 30$ as the number of retrieved elements.

Appendix B. Recommendation effectiveness of tuned methods

B.1. Required recall 0.90

This section shows recommendations provided by the tuned methods optimizing query time and memory usage. Like the previous section, we present two different required recall values.

Fig. B.22 shows recommendations optimizing query time. The methods performed poorly for subsets in general and the *Synthetic* dataset for this required recall value. Overall, the methods provided wrong recommendations for the same datasets and subsets, e.g. *Synthetic* and *MNIST*. On the other hand, looking to the correct recommendations only, the *TMM-S+* provided recommendations close to the optimal or performed better than the Grid Search, mainly for $k = 1$. For $k = 30$, the complete datasets received wrong recommendations from this method, but it generally had the lowest error rate. The *TMM-GS+* had an overall performance more robust, since it provided only a few wrong recommendations.

Regarding memory usage optimization, Fig. B.23 shows an outstanding performance achieved by all methods in general. Here, the datasets *SIFT* and *Synthetic* presented the worst recommendations in general for both k values. The methods tuned with instance generated by a grid search procedure presented a better global performance. They provided fewer wrong recommendations and low error rates when the required recall was not satisfied.

B.2. Required recall 0.99

In opposite to the quick methods, which struggled to provide correct recommendations for this required recall, the tuned methods presented improvements as illustrated in Fig. B.24. This superiority highlights the robustness of these approaches for a requirement more difficult to satisfy. Again, when the *TMM-S+* provide wrong recommendations it presents a low error rate, in general. For correct recommendations all methods perform similarly, but the *TMM-S+* and *TMM-GS+* still perform slightly better, e.g. *Cophir282d* with both k values and *Synthetic* with $k = 30$. The Grid Search is outperformed in most cases for $k = 30$ and performs similarly to the other methods with $k = 1$.

For memory usage optimization in Fig. B.25, the general behavior is the same s for the required recall of 0.90, where *SIFT* and *Synthetic* frequently receive wrong recommendations. Nonetheless, the methods perform notably for the other datasets and overcome the grid search or perform similarly in most cases.

References

[1] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search - The Metric Space Approach, in: Advances in Database Systems, vol. 32, Kluwer, 2006.

[2] G. Navarro, Searching in metric spaces by spatial approximation, VLDB J. 11 (1) (2002) 28–46.

[3] C.T. Jr., A.J.M. Traina, B. Seeger, C. Faloutsos, Slim-trees: High performance metric trees minimizing overlap between nodes, in: EDBT, in: Lecture Notes in Computer Science, vol. 1777, Springer, 2000, pp. 51–65.

[4] M.R. Vieira, C. Traina, F.J. Chino, A.J. Traina, DBM-tree: A dynamic metric access method sensitive to local density data, in: In SBBD, Citeseer, 2004.

[5] L. Chen, Y. Gao, X. Li, C.S. Jensen, G. Chen, Efficient metric indexing for similarity search and similarity joins, IEEE Trans. Knowl. Data Eng. 29 (3) (2017) 556–571.

[6] P. Indyk, R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, in: J.S. Vitter (Ed.), STOC, ACM, 1998, pp. 604–613.

[7] Y. Zhang, K. Huang, G. Geng, C. Liu, Fast kNN graph construction with locality sensitive hashing, in: H. Blockeel, K. Kersting, S. Nijssen, F. Železný (Eds.), ECML PKDD, in: Lecture Notes in Computer Science, vol. 8189, Springer, 2013, pp. 660–674.

[8] J. Wang, H.T. Shen, J. Song, J. Ji, Hashing for similarity search: A survey, 2014, CoRR abs/1408.2927. arXiv:1408.2927.

[9] W. Liu, H. Wang, Y. Zhang, W. Wang, L. Qin, I-LSH: I/O efficient c-approximate nearest neighbor search in high-dimensional space, in: IEEE ICDE, IEEE, 2019, pp. 1670–1673.

[10] O. Jafari, P. Maurya, P. Nagarkar, K.M. Islam, C. Crushev, A survey on locality sensitive hashing algorithms and their applications, 2021, CoRR abs/2102.08942. arXiv:2102.08942.

[11] A. Esuli, Use of permutation prefixes for efficient and scalable approximate similarity search, Inf. Process. Manag. 48 (5) (2012) 889–902.

[12] G. Amato, C. Gennaro, P. Savino, MI-File: using inverted files for scalable approximate similarity search, Multimedia Tools Appl. 71 (3) (2014) 1333–1362.

[13] B. Naidan, L. Boytsov, E. Nyberg, Permutation search methods are efficient, yet faster search is possible, PVLDB 8 (12) (2015) 1618–1629.

[14] D. Novak, P. Zezula, PPP-codes for large-scale similarity searching, Trans. Large Scale Data Knowl. Centered Syst. 24 (2016) 61–87.

[15] K. Figueroa, R. Paredes, N. Reyes, New permutation dissimilarity measures for proximity searching, in: S. Marchand-Maillet, Y.N. Silva, E. Chávez (Eds.), SISAP, in: Lecture Notes in Computer Science, vol. 11223, Springer, 2018, pp. 122–133.

[16] J.W. Jaromczyk, G.T. Toussaint, Relative neighborhood graphs and their relatives, Proc. IEEE 80 (9) (1992) 1502–1517.

[17] G. Navarro, N. Reyes, Dynamic spatial approximation trees for massive data, in: T. Skopal, P. Zezula (Eds.), SISAP, IEEE Computer Society, 2009, pp. 81–88.

[18] R. Paredes, E. Chávez, Using the *k*-nearest neighbor graph for proximity searching in metric spaces, in: M.P. Consens, G. Navarro (Eds.), SPIRE, in: Lecture Notes in Computer Science, vol. 3772, Springer, 2005, pp. 127–138.

[19] Y. Malkov, A. Ponomarenko, A. Logvinov, V. Krylov, Approximate nearest neighbor algorithm based on navigable small world graphs, Inf. Syst. 45 (2014) 61–68.

[20] M. Iwasaki, D. Miyazaki, Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data, 2018, CoRR abs/1810.07355. arXiv:1810.07355.

[21] C. Fu, C. Xiang, C. Wang, D. Cai, Fast approximate nearest neighbor search with the navigating spreading-out graph, Proc. VLDB Endow. 12 (5) (2019) 461–474.

[22] L.C. Shimomura, R.S. Oyamada, M.R. Vieira, D.S. Kaster, A survey on graph-based methods for similarity searches in metric spaces, Inf. Syst. 95 (2021) 101507.

[23] F. Falchi, C. Gennaro, P. Zezula, A content–addressable network for similarity search in metric spaces, in: G. Moro, S. Bergamaschi, S. Joseph, J.-H. Morin, A.M. Ouksel (Eds.), Databases, Information Systems, and Peer-to-Peer Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 98–110.

[24] M. Batko, V. Dohnal, P. Zezula, M-grid: Similarity searching in grid, in: Proceedings of the International Workshop on Information Retrieval in Peer-to-Peer Networks, in: P2PIR '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 17–24, http://dx.doi.org/10.1145/1183579.1183583.

[25] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, H. Zhang, Fast approximate nearest-neighbor search with k-nearest neighbor graph, in: T. Walsh (Ed.), IJCAI, IJCAI/AAAI, 2011, pp. 1312–1317.

[26] M. Aumüller, E. Bernhardsson, A.J. Faithfull, ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms, Inf. Syst. 87 (2020).

[27] R.S. Oyamada, L.C. Shimomura, S.B. Junior, D.S. Kaster, Towards proximity graph auto-configuration: An approach based on meta-learning, in: J. Darmont, B. Novikov, R. Wrembel (Eds.), ADBIS, in: Lecture Notes in Computer Science, vol. 12245, Springer, 2020, pp. 93–107.

[28] R. Paredes, E. Chávez, K. Figueroa, G. Navarro, Practical construction of *k*-nearest neighbor graphs in metric spaces, in: C. Àlvarez, M.J. Serna (Eds.), WEA, in: Lecture Notes in Computer Science, vol. 4007, Springer, 2006, pp. 85–97.

[29] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, J. ACM 45 (6) (1998) 891–923.

[30] W. Dong, M. Charikar, K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures, in: WWW, ACM, 2011, pp. 577–586.

[31] K. Aoyama, K. Saito, T. Yamada, N. Ueda, Fast similarity search in small-world networks, in: S. Fortunato, G. Mangioni, R. Menezes, V. Nicosia (Eds.), CompleNet, in: Studies in Computational Intelligence, vol. 207, 2009, pp. 185–196.

[32] S. Barton, V. Dohnal, J. Sedmidubsky, P. Zezula, Toward self-organizing search systems, in: Computational Social Network Analysis: Trends, Tools and Research Advances, Springer London, London, 2010, pp. 49–80, http://dx.doi.org/10.1007/978-1-84882-229-0_3.

[33] J.M. Kleinberg, The small-world phenomenon: an algorithmic perspective, in: F.F. Yao, E.M. Luks (Eds.), ACM STOC, ACM, 2000, pp. 163–170.

[34] Y.A. Malkov, D.A. Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, IEEE Trans. Pattern Anal. Mach. Intell. 42 (4) (2020) 824–836, http://dx.doi.org/10.1109/TPAMI.2018.2889473.

[35] P. Brazdil, C.G. Giraud-Carrier, C. Soares, R. Vilalta, Metalearning - Applications to Data Mining, in: Cognitive Technologies, Springer, 2009.

[36] K.A. Smith-Miles, Towards insightful algorithm selection for optimisation using meta-learning concepts, in: IJCNN, Part of the IEEE WCCI, IEEE, 2008, pp. 4118–4124.

[37] R. Priya, B.F. de Souza, A.L.D. Rossi, A.C.P. de Leon Ferreira de Carvalho, Using genetic algorithms to improve prediction of execution times of ML tasks, in: HAIS, in: Lecture Notes in Computer Science, vol. 7208, Springer, 2012, pp. 196–207.

[38] C. Lemke, M. Budka, B. Gabrys, Metalearning: a survey of trends and technologies, Artif. Intell. Rev. 44 (1) (2015) 117–130.

[39] R.G. Mantovani, A.L.D. Rossi, J. Vanschoren, B. Bischl, A.C.P.L.F. de Carvalho, To tune or not to tune: Recommending when to adjust SVM hyper-parameters via meta-learning, in: IJCNN, IEEE, 2015, pp. 1–8.

[40] C. Soares, P. Brazdil, P. Kuba, A meta-learning method to select the kernel width in support vector regression, Mach. Learn. 54 (3) (2004) 195–209.

[41] G.F.C. Campos, S.M. Mastelini, G.J. Aguiar, R.G. Mantovani, L.F. de Melo, S.B. Junior, Machine learning hyperparameter selection for contrast limited adaptive histogram equalization, EURASIP 2019 (2019) 59.

[42] Q. Sun, B. Pfahringer, Pairwise meta-rules for better meta-learning-based algorithm ranking, Mach. Learn. 93 (1) (2013) 141–161.

[43] J. Vanschoren, Meta-learning: A survey, 2018, CoRR abs/1810.03548. arXiv:1810.03548.

[44] F. Korn, B.-. Pagel, C. Faloutsos, On the "dimensionality curse" and the "self-similarity blessing", IEEE Trans. Knowl. Data Eng. (2001) 96–111.

[45] M.E. Houle, Local intrinsic dimensionality I: An extreme-value-theoretic foundation for similarity applications, in: C. Beecks, F. Borutta, P. Kröger, T. Seidl (Eds.), SISAP, in: Lecture Notes in Computer Science, vol. 10609, Springer, 2017, pp. 64–79.

[46] M. Aumüller, M. Ceccarello, Benchmarking nearest neighbor search: Influence of local intrinsic dimensionality and result diversity in real-world datasets, in: EDML SDM, in: CEUR Workshop Proceedings, vol. 2436, CEUR-WS.org, 2019, pp. 14–23.

[47] C.C. Aggarwal, A. Hinneburg, D.A. Keim, On the surprising behavior of distance metrics in high dimensional spaces, in: J.V. den Bussche, V. Vianu (Eds.), ICDT, in: Lecture Notes in Computer Science, vol. 1973, Springer, 2001, pp. 420–434.

[48] J.A. Costa, A.O.H. III, Geodesic entropic graphs for dimension and entropy estimation in manifold learning, IEEE Trans. Signal Process. 52 (8) (2004) 2210–2221.

[49] D. François, V. Wertz, M. Verleysen, The concentration of fractional distances, IEEE Trans. Knowl. Data Eng. 19 (7) (2007) 873–886.

[50] E. Levina, P.J. Bickel, Maximum likelihood estimation of intrinsic dimension, in: NIPS, 2004, pp. 777–784.

[51] A.C. Lorena, L.P.F. Garcia, J. Lehmann, M.C.P. de Souto, T.K. Ho, How complex is your classification problem?: A survey on measuring classification complexity, ACM Comput. Surv. 52 (5) (2019) 107:1–107:34.

[52] K. Lin, H.V. Jagadish, C. Faloutsos, The TV-tree: An index structure for high-dimensional data, VLDB J. 3 (4) (1994) 517–542.

[53] S. Berchtold, D.A. Keim, H. Kriegel, The X-tree : An index structure for high-dimensional data, in: T.M. Vijayaraman, A.P. Buchmann, C. Mohan, N.L. Sarda (Eds.), VLDB, Morgan Kaufmann, 1996, pp. 28–39.

[54] K. Chakrabarti, S. Mehrotra, Local dimensionality reduction: A new approach to indexing high dimensional spaces, in: A.E. Abbadi, M.L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, K. Whang (Eds.), VLDB, Morgan Kaufmann, 2000, pp. 89–100.

[55] K. Echihabi, K. Zoumpatianos, T. Palpanas, H. Benbrahim, Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search, Proc. VLDB Endow. 13 (3) (2019) 403–420, http://dx.doi.org/10.14778/3368289.3368303.

[56] Z. Zhou, S. Tan, Z. Xu, P. Li, Möbius transformation for fast inner product search on graph, in: H.M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E.B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 8216–8227, http://dx.doi.org/10.5555/3454287.3455025.

[57] W. Zhao, S. Tan, P. Li, SONG: Approximate nearest neighbor search on GPU, in: 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020, IEEE, 2020, pp. 1033–1044, http://dx.doi.org/10.1109/ICDE48307.2020.00094.

[58] J. Augustine, S. Shetiya, M. Esfandiari, S.B. Roy, G. Das, A generalized approach for reducing expensive distance calls for a broad class of proximity problems, in: G. Li, Z. Li, S. Idreos, D. Srivastava (Eds.), SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, ACM, 2021, pp. 142–154, http://dx.doi.org/10.1145/3448016.3457303.

[59] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, X. Lin, Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0), 2016, CoRR abs/1610.02455. arXiv:1610.02455.

[60] M. Wang, X. Xu, Q. Yue, Y. Wang, A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search, Proc. VLDB Endow. 14 (11) (2021) 1964–1978, URL http://www.vldb.org/pvldb/vol14/p1964-wang.pdf.

[61] M. Muja, D.G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: VISAPP, INSTICC Press, 2009, pp. 331–340.

[62] X. Zhou, C. Chai, G. Li, J. SUN, Database meets artificial intelligence: A survey, IEEE Trans. Knowl. Data Eng. (2020) 1, http://dx.doi.org/10.1109/TKDE.2020.2994641.

[63] D. Baranchuk, A. Babenko, Towards similarity graphs constructed by deep reinforcement learning, 2019, CoRR abs/1911.12122. arXiv:1911.12122.

[64] D. Baranchuk, D. Persiyanov, A. Sinitsin, A. Babenko, Learning to route in similarity graphs, in: K. Chaudhuri, R. Salakhutdinov (Eds.), ICML, in: Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 475–484.

[65] C. Li, M. Zhang, D.G. Andersen, Y. He, Improving approximate nearest neighbor search through learned adaptive early termination, in: D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, H.Q. Ngo (Eds.), SIGMOD, ACM, 2020, pp. 2539–2554.

[66] M. Antol, J. Olha, T. Slaninákóvá, V. Dohnal, Learned metric index - proposition of learned indexing for unstructured data, Inf. Syst. 100 (2021) 101774.

[67] T. Slaninákóvá, M. Antol, J. Olha, V. Kana, V. Dohnal, Data-driven learned metric index: An unsupervised approach, in: N. Reyes, R. Connor, N.M. Kriege, D. Kazempour, I. Bartolini, E. Schubert, J. Chen (Eds.), Similarity Search and Applications - 14th International Conference, SISAP 2021, Dortmund, Germany, September 29 - October 1, 2021, Proceedings, in: Lecture Notes in Computer Science, vol. 13058, Springer, 2021, pp. 81–94, http://dx.doi.org/10.1007/978-3-030-89657-7_7.

[68] M. Hünemörder, P. Kröger, M. Renz, Towards a learned index structure for approximate nearest neighbor search query processing, in: N. Reyes, R. Connor, N.M. Kriege, D. Kazempour, I. Bartolini, E. Schubert, J. Chen (Eds.), Similarity Search and Applications - 14th International Conference, SISAP 2021, Dortmund, Germany, September 29 - October 1, 2021, Proceedings, in: Lecture Notes in Computer Science, vol. 13058, Springer, 2021, pp. 95–103, http://dx.doi.org/10.1007/978-3-030-89657-7_8.

[69] M. Aumüller, M. Ceccarello, The role of local dimensionality measures in benchmarking nearest neighbor search, Inf. Syst. 101 (2021) 101807.

[70] C.B. Barber, D.P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Trans. Math. Softw. 22 (4) (1996) 469–483.

[71] M. Ester, H. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: E. Simoudis, J. Han, U.M. Fayyad (Eds.), KDD, AAAI Press, 1996, pp. 226–231.

[72] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, F. Rabitti, CoPhIR: A test collection for content-based image retrieval, 2009, Computing Research Repository abs/0905.4627v2.

[73] L. Boytsov, B. Naidan, Engineering efficient and effective non-metric space library, in: SISAP, in: Lecture Notes in Computer Science, vol. 8199, Springer, 2013, pp. 280–293.