Universidade Estadual de Londrina

RAFAEL SEIDI OYAMADA

A META-LEARNING APPROACH FOR AUTO-SELECTION AND AUTO-CONFIGURATION OF PROXIMITY GRAPHS

Londrina
2021

RAFAEL SEIDI OYAMADA

A META-LEARNING APPROACH FOR AUTO-SELECTION AND AUTO-CONFIGURATION OF PROXIMITY GRAPHS

A thesis presented to the Graduate Program in Computer Science at the State University of Londrina to obtain the degree of Master of Science in Computer Science.

Advisor: Prof. Dr. Daniel dos Santos Kaster

Londrina
2021

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Oyamada, Rafael Seidi.
A Meta-Learning Approach for Auto-Selection and Auto-Configuration of Proximity Graphs / Rafael Seidi Oyamada. - Londrina, 2021.
94 f. : il.

Orientador: Daniel dos Santos Kaster.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2021.
Inclui bibliografia.

1. Banco de Dados - Tese. 2. Buscas por Similaridade - Tese. 3. Grafos de Proximidade - Tese. 4. Meta-Aprendizado - Tese. I. Kaster, Daniel dos Santos . II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 519

RAFAEL SEIDI OYAMADA

A META-LEARNING APPROACH FOR AUTO-SELECTION AND AUTO-CONFIGURATION OF PROXIMITY GRAPHS

A thesis presented to the Graduate Program in Computer Science at the State University of Londrina to obtain the degree of Master of Science in Computer Science.

EXAMINATION BOARD

_____

Advisor: Prof. Dr. Daniel dos Santos Kaster
State University of Londrina – UEL

_____

Prof. Dr. Renato Bueno
Universidade Federal de São Carlos - UFSCAR

_____

Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina – UEL

Londrina, 16 de abril, 2021.

Este trabalho é dedicado às crianças adultas
que, quando pequenas, sonharam em se
tornar cientistas.

ACKNOWLEDGEMENTS

I would like to thank the National Council for Scientific and Technological Development (CNPq) and the National Council for the Improvement of Higher Education (CAPES) for the financial support, and the State University of Londrina.

Also, I would like to thank my advisor, Professor Daniel dos Santos Kaster, for the guidance along this journey, Professor Sylvio Barbon Jr. for providing valuable ideas, and my friend Larissa Shimomura for supporting me at the beginning of this master's program.

Furthermore, I am very grateful to have met many special people during my journey at this university. Friends, having you in my life is such a blessing.

Finally, Grandma, without you, none of this would have happened. Thank you very much and know that I fight every day of my life to make you proud.

It is our choices that show what we truly are, far more than our abilities.

OYAMADA, R. S. **Uma Abordagem de Meta-Aprendizado para Auto-Seleção e Auto-Configuração de Grafos de Proximidade.** 2021. 93 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2021.

RESUMO

Devido à alta produção de dados complexos, as últimas décadas proporcionaram um avanço considerável no desenvolvimento de métodos de busca por similaridade. Esses métodos consistem em indexar e recuperar dados por meio de suas características intrínsecas. Recentemente, os métodos baseados em grafos superaram outros tipos de métodos na literatura de buscas por similaridade aproximada, tais como os métodos baseados em árvore, permutação e hash. No entanto, encontrar um grafo adequado junto com seus parâmetros é uma tarefa desafiadora e demorada. Uma vez que não existe uma parametrização precisa que se adapte à maioria dos conjuntos de dados e exigências do usuário, a escolha dos parâmetros é arbitrária ou baseada em uma busca em grade de execuções. O objetivo principal deste trabalho é desenvolver uma abordagem inteligente baseada em técnicas de meta-aprendizado para recomendar uma configuração de grafo adequada para um determinado conjunto de dados. Nossas principais contribuições são a proposta de uma arquitetura genérica e estratégias para construir um sistema de recomendação preciso, evitando experimentações exaustivas para parametrizar métodos baseados em grafos para buscas por similaridade. Especificamente, esta dissertação de mestrado detalha a coleta de conhecimento para resolver o problema e duas estratégias diferentes para instanciar um recomendador. A primeira estratégia é uma abordagem global, que induz modelos de regressão em relação a todo o espaço do conhecimento. O segundo é baseado na similaridade do conjunto de dados, aprendendo modelos de regressão para grupos de conjuntos de dados com propriedades semelhantes. Afirmamos que particionar o espaço do conjunto de dados para aprendizagem melhora a precisão das recomendações. Este trabalho emprega uma variedade de conjuntos de dados reais com características extraídas de imagens e um grande conjunto de conjuntos de dados sintéticos variando os principais parâmetros que afetam a recuperação de similaridade. Apresentamos experimentos avaliando as estratégias de instanciação propostas que atestam que nossas abordagens superam as linhas de base na maioria dos casos. Também discutimos experimentos explorando aspectos das técnicas propostas para apoiar as alternativas que escolhemos ao longo do trabalho. Os resultados mostram que nossas propostas fornecem recomendações adequadas para grafos de proximidade, auxiliando os usuários na construção de índices eficientes para recuperação por similaridade.

Palavras-chave: grafos de proximidade; buscas por similaridade; meta-aprendizado.

OYAMADA, R. S. **A Meta-Learning Approach for Auto-Selection and Auto Configuration of Proximity Graphs.** 2021. 93 p. Thesis (Master of Science in Computer Science) – State University of Londrina, Londrina, 2021.

ABSTRACT

Due to the high production of complex data, the last decades have provided a considerable progress in developing similarity search methods. Such methods consist of indexing and retrieving data through their intrinsic characteristics. Recently, graph-based methods have outperformed other types of methods in the literature of approximate similarity search, such as the tree-, permutation-, and hash-based methods. However, finding a suitable graph along with its parameters is a challenging and time-consuming task. Since there is no precise parameterization that suits most datasets and user constraints, the choice of parameters is either arbitrary or based on a grid search of executions. The main objective of this work is to develop an intelligent approach based on meta-learning techniques to recommend a suitable graph configuration for a given dataset. Our main contributions are the proposal of a generic architecture and strategies to build an accurate recommendation system avoiding exhaustive experimentation to parameterize graph-based methods for similarity searching. Specifically, this master's thesis details the gathering of knowledge to address the problem and two different strategies to instantiate a recommender. The first strategy is a global approach, which induces regression models regarding the whole knowledge space. The second one is based on dataset-similarity, learning regression models for clusters of datasets with similar properties. We claim that partitioning the dataset space for learning improves the accuracy of the recommendations. This work employs an assortment of real datasets with features extracted from images and a large set of synthetic datasets varying the main parameters affecting similarity retrieval. We present experiments evaluating the proposed instantiation strategies that attest that our approaches outperform the baselines in most cases. We also discuss experiments exploring aspects of the proposed techniques to support the alternatives we chose throughout the work. The results show that our proposals provide suitable recommendations for proximity graphs, assisting users in building efficient indexes for similarity retrieval.

Keywords: proximity graphs; similarity searching; meta-learning.

LIST OF FIGURES

Figure 1 – A practical example of retrieving elements based on their similarities. . 19
Figure 2 – Shapes of the Lp norms L_1, L_2 and L_∞ in the space formed by the elements that are equidistant to the center element u regarding each norm. 21
Figure 3 – (a) *Range* query and (b) *k-NN* query. 21
Figure 4 – The middle third Cantor set (adapted from [1]). 27
Figure 5 – (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. 34
Figure 6 – The behavior of graph-based methods for the dataset Texture and increasing *NN*. a) Smallest number of restarts for the each graph and *NN* value. b) Query time for the corresponding configurations of the plot on the left. 35
Figure 7 – The proposed framework, illustrating how the overall process of building the meta-dataset, instantiating and using the meta-model, and generating recommendations according to inputs provided by a user. . 43
Figure 8 – A global instantiation of meta-models following the framework in Figure 7. There is a meta-model for each of the n meta-targets. 50
Figure 9 – The importance rate of each meta-feature per target (recall or query time) and category: (a) graph configurations, (b) general and information-theoretical, and (c) statistical. 53
Figure 10 – Comparison of the recommendations provided by the methods attending to the requirement of the lowest query time. 56
Figure 11 – Comparison of the recommendations provided by the methods attending to the requirement of least memory usage. 57
Figure 12 – Accuracy of the predictions of the meta-models per recall interval for each dataset. 58
Figure 13 – The components and task flow of the proposed dataset-similarity-based recommender instantiation. 62
Figure 14 – Complexity meta-features that improve the measurement of hardness of datasets. This example regards all real datasets employed in this work for a fixed graph type and fixed k value. In the left we fixed $NN = 5$ and in the right $R = 5$. The first line refers to the intrinsic dimensionality, and the higher its value the harder the dataset is. The second line refers to the relative variance, and the lower its value the harder the dataset is. 65

Figure 15 – Interpolation of a meta-target to generate weak-labeled meta-instances. The original values for the recall (left) and interpolated parameter values (right), for varying NN and R. 67
Figure 16 – Performance comparison using train/test procedures by alternating between the original and the augmented meta-datasets. 68
Figure 17 – Enhancement promoted by adding new meta-features and/or meta-instances from new datasets. 70
Figure 18 – (Meta-)Feature selection method considering the eps selected by the previous section. 72
Figure 19 – The meta-features defining each meta-target's dataset space according to the RF method and their importances. 72
Figure 20 – r^2 scores achived by each dataset-similarity-based meta-model, regarding each meta-target and each dataset. 74
Figure 21 – Predictive performance distribution regarding each method and meta-target for all datasets. 75
Figure 22 – The elapsed time for providing the final predictions for all meta-targets according to each method and dataset. 75
Figure 23 – Recommendations optimizing the query time regarding the quick methods. 77
Figure 24 – Recommendations optimizing the number of distance computations regarding the quick methods. 78
Figure 25 – Recommendations optimizing memory usage regarding the quick methods. This metric is estimated by the number of edges (NN parameter) of the graph. The higher this value, the higher the memory consumption. 79
Figure 26 – Recommendations optimizing the query time regarding the tuned methods. 80
Figure 27 – Recommendations optimizing the number of distance computations regarding the tuned methods. 81
Figure 28 – Recommendations optimizing memory usage regarding the tuned methods. 81

LIST OF TABLES

Table 1 – Complexity metrics employed to measure the hardness of a dataset specifically for similarity search problems. 29
Table 2 – All attributes of the final meta-dataset built in this work. 46
Table 3 – Available optimization goals. 49
Table 4 – The real and synthetic datasets employed and their characteristics. . . . 52
Table 5 – Relative performances of the generic and tuned meta-models. 55
Table 6 – Recommendations by GMM that satisfy the requirements of *recall* $\geq$ 0.90 with the lowest *query time*. 59
Table 7 – The real and synthetic datasets employed in this chapter and their characteristics. 69
Table 8 – Cluster information on dataset spaces defined by the feature selection methods. 71

LIST OF ABBREVIATIONS AND ACRONYMS

AI4DB Artificial Intelligence for Databases

ANN Approximate Nearest Neighbor

DBMS Database Management System

DRL Deep Reinforcement Learning

GMM Generic Meta-Model

GNNS Graph Nearest Neighbor Search

ID Intrinsic Dimensionality

LID Local Intrinsic Dimensionality

k-NN k-Nearest Neighbors

k-NNG k-Nearest Neighbor Graph

LSH Locality Sensitive Hashing

ML Machine Learning

NN Nearest Neighbors

NSW Navigable Small World

PCA Principal Component Analysis

R Restarts

RC Relative Contrast

RF Random Forest

RV Relative Variation

TMM-GS Tuned Meta-Model with Grid Search

TMM-S Tuned Meta-Model with Subsets

CONTENTS

1 INTRODUCTION 15
1.1 Contributions 17
1.2 Outline 18

2 FUNDAMENTAL CONCEPTS 19
2.1 Similarity Search 19
2.1.1 Similarity Queries 20
2.1.2 Indexing Methods for Similarity Searching 22
2.2 Proximity Graphs for Similarity Search 23
2.2.1 Searching in Proximity Graphs 23
2.2.2 Graph-based Indexing Methods 24
2.3 Dataset Metrics for Similarity Retrieval 26
2.4 Meta-Learning 29
2.4.1 Gathering Meta-knowledge 30

3 THE PROBLEM OF PARAMETER RECOMMENDATION FOR GRAPH-BASED INDEXING METHODS 33
3.1 The Impact of Parameters for Graph-based Methods 33
3.2 Parameter Recommendation for Database Systems 36
3.3 Parameter Recommendation for Graph-based Methods 38

4 A META-LEARNING APPROACH FOR AUTO-CONFIGURATION OF PROXIMITY GRAPHS 41
4.1 Modeling Proximity Graph Recommendation as a Meta-learning Problem 41
4.2 A Framework to Provide Recommendations 42
4.3 Strategy for Gathering Meta-Knowledge 44
4.3.1 Meta-features 44
4.3.2 Meta-targets 45
4.4 Meta-learning and Recommendation 47
4.4.1 Definition of Meta-Models 48
4.4.2 Recommendation 48

5 A GLOBAL META-LEARNING RECOMMENDER INSTANTIATION 50
5.1 Overview of the Instantiation 50
5.2 Experimental setup 51

5.2.1 Datasets . . . 51
5.2.2 Dataset Characterization . . . 51
5.2.3 Performance Measurement . . . 52
5.2.4 Learning Method . . . 53
5.3 Experimental Results . . . 53
5.3.1 Analysis of Meta-feature Importance . . . 53
5.3.2 Prediction Accuracy of the Meta-models . . . 54
5.3.3 Recommendation Effectiviness . . . 55

6 A DATASET-SIMILARITY-BASED META-LEARNING RECOMMENDER INSTANTIATION . . . 60
6.1 Instantiation Overview . . . 61
6.1.1 Meta-Feature Selection . . . 62
6.1.2 Dataset Clustering . . . 62
6.1.3 Meta-model Inference . . . 64
6.2 Improvements to the Meta-Knowledge . . . 64
6.2.1 New Complexity Meta-Features . . . 64
6.2.2 Data Augmentation based on Interpolation . . . 66
6.3 Analysis of the Instantiation's Supporting Techniques . . . 66
6.3.1 Impact of the Data Augmentation Technique . . . 67
6.3.2 Impact of the New Meta-features and Datasets . . . 68
6.3.3 Impact of Feature Selection Techniques on the Dataset-similarity-based Meta-Models . . . 70
6.4 Analysis of the Instantiation's Prediction Accurracy and Execution Time . . . 73
6.4.1 Prediction Accuracy of the Dataset-Similarity-based Meta-Models . . . 73
6.4.2 Prediction Accuracy and Recommendation Time of the Global and Dataset-Similarity-based Methods . . . 74
6.5 Analysis of the Recommendation Effectiveness . . . 76
6.5.1 Quick Methods . . . 76
6.5.2 Tuned Methods . . . 79

7 CONCLUSION . . . 83
7.1 Discussion . . . 83
7.2 Future Work . . . 85

REFERENCES . . . 86

PUBLICATIONS . . . 93

1 INTRODUCTION

Complex data (images, long texts, audios, etc.) are common in pattern recognition, image retrieval, data mining, and other tasks. In general, complex data are represented through feature vectors composed of measures and properties extracted from the intrinsic content of the data and retrieved using dissimilarity relations between pairs of feature vectors [2]. 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 [3].

In the literature, there are several access methods suitable for indexing complex data. These methods can be divided into four groups: tree-based [4, 5, 6, 7], hashing-based [8, 9, 10, 11, 12], permutation-based [13, 14, 15, 16, 17], and *graph-based* [18, 19, 20, 21, 22, 23]. 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). In this work, 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 [24, 15, 20].

Widely used graph-based methods, such as the k-NNG [19] and the NSW [20], 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 to be traversed by the query algorithms towards the elements that are the most similar to the query element. However, a large number of neighbors also increases the memory footprint and the cost of vertex expansion during the search, as it is proportional to the size of the vertex's adjacency. Additionally, depending on the graph structure, the query algorithm may not be able to find a path from the search source vertex to every vertex that is part of the answer. A common approach to alleviate this problem is to execute a given number of traversals, starting each execution from a different source 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 [25, 23]. Aumüller et al. [25] proposed a benchmark to understand better approximate nearest neighbor (ANN) algorithms, including graph-based methods. Ne-

vertheless, the experimental results using different datasets 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 [23]. Our results indicated that neither a graph type is the best for all cases nor is there a universally suitable parameterization for any graph type. The results also showed that there are trade-offs between construction and search algorithms according to their parameters. Although these works identified general patterns useful to guide the choice of the type of graph and its parameters, these patterns are not easily interpretable, leading to suboptimal parameterizations. A common approach to define the graph configuration is to perform exhaustive evaluations using a grid search considering combinations of graph-based methods and their parameters. However, this is time-consuming and limited to the tested combinations.

Several studies aimed at understanding which characteristics of a dataset significantly influence nearest neighbor algorithms' performance and retrieval quality. In the similarity search community, one classical reason for this issue is the "curse of dimensionality", which says that it is harder to perform nearest neighbor searches in high-dimensional spaces [26, 27, 28]. Korn et al. [29] claimed that the intrinsic dimensionality of a dataset is the most crucial factor impacting the search performance. The intrinsic dimensionality is the theoretical minimum number of dimensions d that a dataset embedded into a e-dimensional space, $d \leq e$, can be described maintaining its intrinsic characteristics so that the information loss is negligible [29]. Thus, several authors have employed this metric as a complexity measure for the nearest neighbor problem [30, 31, 32, 33]. Furthermore, the literature presents other metrics with similar purposes. For instance, the fractal correlation [26], the relative variance [34], the relative contrast [35], and the clustering coefficient extracted from a *k-NN* graph [36].

This work is based on the premise that it is possible to learn the relation between dataset metrics 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 using a particular index configuration would perform on a dataset and provide a proper recommendation for the case. This idea is known in the machine learning community as meta-learning, whose overall objective is to characterize the learning outcomes for previous problems to guide the learning process for a new problem. Meta-learning is essentially the notion of "learning to learn", wherein a meta-model learns how to learn the solution of an unknown problem [37, 38, 39]. This concept has been applied over different areas, such as time series algorithm selection [40], deciding if an algorithm should be tuned or not [41], clustering algorithm selection [42], recommending image segmentation algorithms [43], among others.

1.1 Contributions

Considering the relevance of graph-based methods nowadays and the recent success of meta-learning approaches, the objective of this master's thesis is to develop an intelligent system to recommend suitable configurations of proximity graphs for similarity retrieval. This work addresses the difficulty of finding suitable trade-offs between construction and searching parameters for these graphs for a specific dataset and user requirements.

We design our solution based on meta-learning techniques and focus on similarity searching on image databases and three of the most widely used base graph-based method types to execute approximate queries. Our proposed meta-models relate dataset characteristics, graph-based method types, query and indexing parameters, and search performance (e.g., execution time and query accuracy). The recommendation is a combination of graph-based method type and construction and query parameters expected to achieve the requested query accuracy in the shortest time or consuming the smallest amount of memory for the given dataset. We summarize our contributions as follows.

- Proposal of a general framework based on meta-learning for parameterizing graph-based methods for similarity searches.
- Gathering of knowledge for the problem, including identifying relevant features to describe datasets for similarity retrieval and building a rich meta-dataset considering three types of graph-based methods, varying their construction and search parameters, on several real and synthetic datasets.
- Proposal of two strategies to instantiate a recommender: (i) a global meta-learning approach, which learns meta-models considering the whole dataset space; and (ii) a dataset-similarity-guided meta-learning approach, which induces models on partitions of the dataset space that cluster datasets that have similar properties and chooses the meta-model from the partition that best fit the input dataset to perform the recommendation.
- Implementation of recommenders following the two proposed instantiation strategies, including variations that employ fine-tuning of the meta-models using instances extracted from the input dataset during the recommendation time.
- Extensive experimental evaluation of the developed recommenders comparing them to baselines and analyzing the behavior of our solutions' main building blocks.

We compared our proposed recommenders with two baselines: the best general setup, which is the parameterization that achieves the best performance for most datasets, and the grid search procedure, which is the most adopted method for selecting

graph algorithms and their parameters. The recommender instantiations trained using the collected meta-database consistently provide superior parameter settings for unseen datasets than the best general setup. In general, the recommendations are comparable to the ones provided by the grid search approach but demanding 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 quite close to the optimal one.

1.2 Outline

This work is organized as follows. Chapter 2 introduces an overview of similarity searches, focusing on proximity graphs, and describes metrics that measure how complex a specific similarity search problem is. Also, we present the main concepts of meta-learning for understanding our proposal. Subsequently, in Chapter 3, we describe the problem of finding suitable graph-based methods and their parameter values for similarity searching. We also discuss related works and approaches employing machine learning techniques for solving similar problems in other areas of database management. Next, Chapter 4 describes the general framework we propose to recommend graph configurations for similarity searching, modeled as a meta-learning problem. Subsequently, we present the two instantiation strategies of our framework: the global meta-learning approach, published in [44] (Chapter 5), and the dataset-similarity-guided meta-learning strategy, which surpasses the global one regarding the predictive performance (Chapter 6). Finally, we conclude this work and mention further research in Chapter 7.

2 FUNDAMENTAL CONCEPTS

This chapter begins by introducing similarity searching (section 2.1) and surveys the data structures employed in this area (subsection 2.1.2). section 2.2 presents proximity graphs and the main graph-based methods for similarity searching, which are the focus of this work. Subsequently, section 2.3 describes the fundamental metrics used for measuring the nearest neighbor complexity. Lastly, section 2.4 presents the main meta-learning concepts to understand this thesis.

2.1 Similarity Search

In the context of this work, complex data refer to data containing rich content, such as images, videos, and audios. Complex data are usually represented by feature vectors extracted from the data content. Feature vectors allow users to measure the similarity between pairs of complex data. 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 $<$, $>$, $\leq$, $\geq$. Similarity-based systems have been used in several areas, including pattern recognition, anomaly detection, recommendation systems, data mining, and others.

Similarity search retrieves similar elements from a database given one or more reference elements according to their features. Figure 1 illustrates this idea. The set of feature vectors and the similarity measure employed, typically a distance function, compose the so-called similarity space. A similarity query comprises one or more similarity-based query operators. It returns all elements in the similarity space that satisfy the (sequence of) operator(s), usually relying on indexing structures. The following sections introduce the most common similarity queries and types of indexing methods for similarity searching.

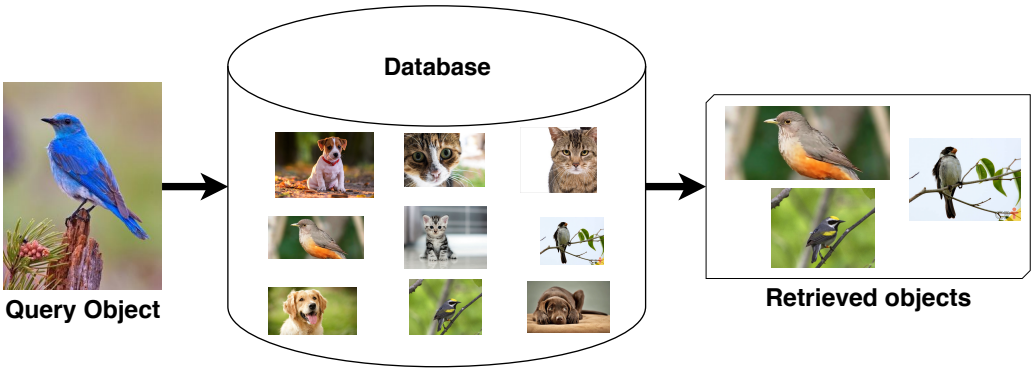


Figure 1 – A practical example of retrieving elements based on their similarities.

2.1.1 Similarity Queries

The similarity between a pair of complex data is usually computed by a distance function δ that quantifies the (dis)similarity between the corresponding pair of feature vectors. The smaller the value returned by this function, the more similar the compared vectors. A set of feature vectors and a distance function compose a similarity space commonly modeled as a metric space. A metric space is an unordered pair $\langle \mathbb{S}, \delta \rangle$, where $\mathbb{S}$ is a data domain, and $\delta : \mathbb{S} \times \mathbb{S} \to \mathbb{R}$ is a distance function that satisfies the following properties, $\forall x, y, z \in \mathbb{S}$ [2]:

- Identity: $\delta(x, y) = 0 \Leftrightarrow x = y$;
- Non-negativity: $\delta(x, y) \geq 0$;
- Symmetry: $\delta(x, y) = \delta(y, x)$;
- Triangle inequality: $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$.

There are different types of distance functions, such as the Minkowski distances, the quadratic form distance, the Jaccard's coefficient, the Hausdorff distance, and others. The Minkowski family's distances are the most widely used functions, also called L_p norms. The L_p norms are defined on n-dimensional vectors of real numbers according to the Equation 2.1:

$$\delta(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{2.1}$$

for any $1 \leq p \leq \infty$. The most known metrics from this family are the L_1 (Manhattan or City-Block distance), which computes the distance between two points in a grid-like path; the L_2 (Euclidean distance), which computes the distance between two points following a straight line; and the L_∞ (Chebyshev or Infinity distance) is the maximum difference between the coordinates of the points. Figure 2 illustrates the shapes of these distances in the space given a center element u. The shapes refer to the points whose distance to u is ξ, regarding each metric.

With a similarity space defined, it is possible to retrieve data from it through similarity queries. The two basic types of similarity queries are the *Range* query (*Rq*) and the k-Nearest Neighbor query (k-*NNq*). 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 on 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

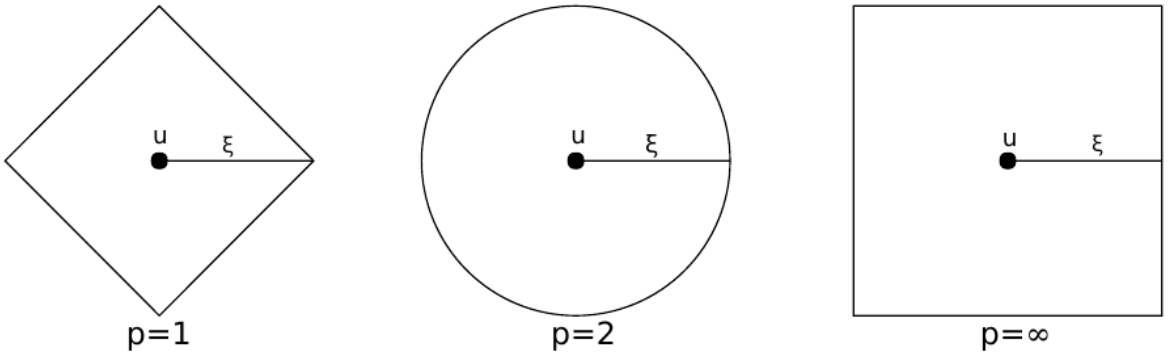
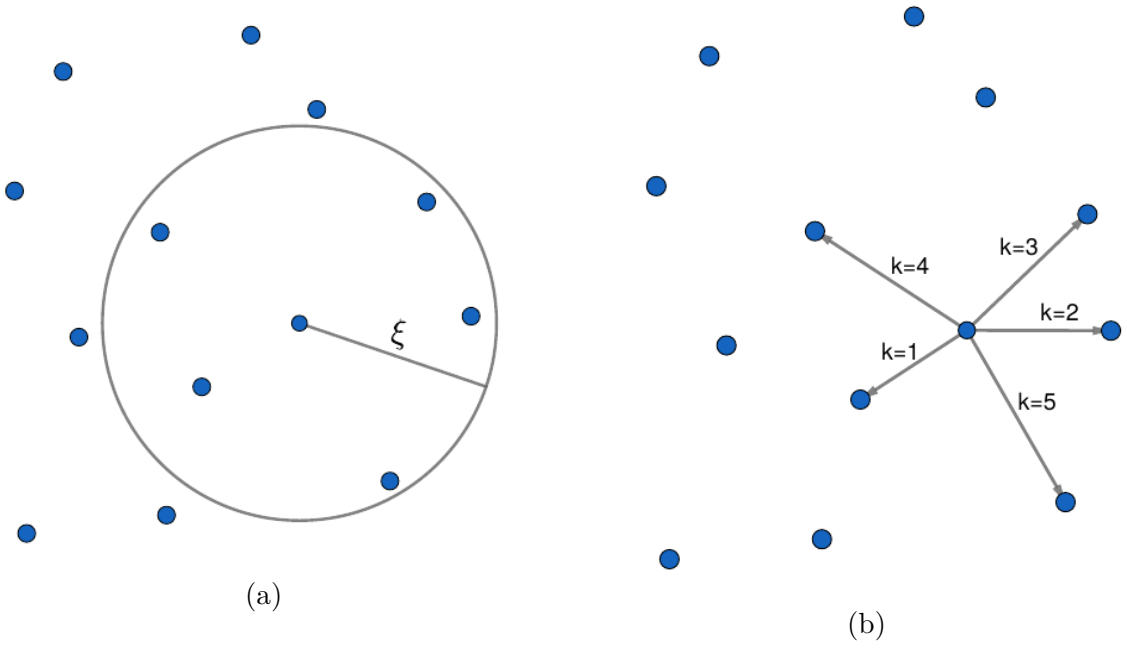


Figure 2 – Shapes of the Lp norms L_1, L_2 and L_∞ in the space formed by the elements that are equidistant to the center element u regarding each norm.

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:

$$kNNq(\delta, s_q, k) = \{s_i \in S \mid |S_k| = k \ \wedge \ \forall s_k \in S_k, \forall s_j \in S - S_k, \delta(q, s_k) \leq \delta(q, s_j)\}$$

Figure 3 shows examples of the *Range* query and the *k-NN* query. In Figure 3(a), the *Range* query returns all the elements inside the circle, while in Figure 3(b) the $k = 5$ nearest neighbors returned are indicated by lines connecting them to the query element.



(a) (b)

Figure 3 – (a) *Range* query and (b) *k-NN* query.

2.1.2 Indexing Methods for Similarity Searching

Due to the large volume of complex databases, and computationally expensive comparisons between elements through distance functions, similarity searches have a high computational cost. Thus, employing suitable methods to index data is very important. Traditional indexing methods included in Database Management Systems (DMBS), such as the B^+-tree and hashing methods, do not apply to complex data because the total order relation does not hold for feature vectors. Hence, several indexing structures for complex data have been proposed, particularly using metric spaces to represent the similarity space. They can be divided into four major groups: tree-based [4], hashing-based [8], permutation-based [14], and graph-based methods [23].

Tree-based methods partition the search space hierarchically into subgroups, allowing to answer a query accessing only a few of them. Searching on trees is a recursive process that starts from the root and iteratively determines which subtrees should be traversed. Literature provides a wide range of tree structures for similarity search, such as DBM-tree [5], FLANN [45], and MRPT [46]. Tree-based methods usually allow to provide exact answers to similarity queries; however, their execution time can be prohibitive for large datasets. Other types of methods focus on approximate retrieval, relaxing the exactness of the results to promote faster query execution, as follows.

In contrast to tree-based indexing techniques, the hash-based methods ideally provide direct access to elements. According to Wang et al. [10], there are two main strategies to perform similarity searches using hashing: indexing data items using hash tables (i.e., store items with the same hash code in the same hash bucket) and using hash codes to approximate the distance between elements. A well-known hashing algorithm for approximate similarity search is the Locality-sensitive Hashing (LSH) [8].

In permutation-based methods, each element is represented as a permutation of a set of references (permutants) sorted by the element's distance. The similarity between elements is based on the distance between their respective permutations. Methods that use permutations include the MI-File [14] and the PP-Index [13]. These methods have achieved good performances, even for datasets with high dimensionality. Graph-based methods map the dataset into a graph. In general, each element is represented by a vertex, and the similarity relationships between pairs of vertices are represented as edges. Graph-based methods include *k-NNG* [19] and *NSW* [20]. Moreover, recent works show that these methods have outperformed the methods of other types in time and/or accuracy of approximate retrieval, especially for high-dimensional datasets. Taking this into consideration, we dedicate the following section to present more details about these methods.

2.2 Proximity Graphs for Similarity Search

A graph is defined as $G = (V, E)$, where V is the set of vertices and E is the set of edges that connects pairs of vertices in V. The most common categories of graphs employed for similarity searching are the proximity graphs [19, 47]. A proximity graph is a graph 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, which is called neighborhood criterion. The neighborhood criterion defines the type of the proximity graph. Usually, such a property represents the similarity between a pair of vertices and can be measured by a distance function $\delta(u, v)$.

The next section introduces the fundamental strategy for similarity searching on proximity graphs and a well-known greedy algorithm based on this strategy. Then, subsection 2.2.2 presents the main base graph-based methods for similarity retrieval proposed in the literature.

2.2.1 Searching in Proximity Graphs

The spatial approximation, introduced by Navarro [3], 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 the return of exact answers to similarity queries on proximity graphs under certain conditions. Given a metric space $\langle S, \delta \rangle$ 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 Equation 2.2 to correctly answer similarity queries using a search algorithm based on the spatial approximation approach, for any query object $q \in S$.

$$\forall u \in V,\ if\ \forall u \in N(u), \delta(q, u) \leq \delta(q, v),\ then\ \forall v' \in V, \delta(q, u) \leq (q, v') \tag{2.2}$$

The main limitation of the spatial approximation to produce exact answers is that the required conditions are too narrow. The only proximity graph such that Equation 2.2 is known to hold is the Delaunay Graph (DG) [48]. However, the DG quickly approximates to a complete graph as either the dataset or its dimensionality increase [23]. Paredes et al. [19] proposed an exact algorithm for *Range* and *k-NN* queries using the spatial approximation on a *k-NNG*. However, their approach is limited to metric spaces and

relies on shortest-path routines during the search, deeply impacting the execution time. Therefore, the spatial approximation has been employed in several works in the literature to produce approximate answers to similarity queries. 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 to select initial vertices and use 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*) is an effective algorithm that executes multiple greedy searches based on spatial approximation and aggregates the partial results into the final result [24]. The *GNNS* can be considered a seminal algorithm as other proposals employ similar ideas. In the *GNNS*, 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.

The choice of bad initial vertices can complicate the search process if the starting vertex is too far from the result. In many algorithms, including the *GNNS*, this choice is random. Nonetheless, an alternative to smooth such a limitation is to define *seed nodes* as starting vertices. These seed nodes can be chosen by random sampling, using a second index structure [49], or selecting well-separated objects in a dataset [50].

2.2.2 Graph-based Indexing Methods

Among the graph-based methods for similarity searches, the k-Nearest Neighbor Graph (*k-NNG*) [51, 24] and the Navigable Small-World graph (*NSW*) [20] are two essential types of graphs. The *k-NNG* [50] 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 regarding 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 that are 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 (e.g., [52]), and methods that relax the connectivity of every element to its k-nearest neighbors (e.g., [53]).

The brute-force construction of the *k-NNG* has a quadratic computational cost, which is unfeasible for large datasets [51, 52]. A preeminent method that generates an approximated version of a *k-NNG* is the *NN-Descent* [52]. The main supporting idea of the *NN-Descent* is that "the neighbor of a neighbor is probably a neighbor". This idea is

used to guide 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 *NN-Descent*'s authors claim that its computational cost is $O(n^{1.14})$, based on empirical results. This cost is achieved through optimizations using local joins, sampling, and early termination. At each iteration, the number of new vertices and edges to be inserted in the *k-NN* decrease, as the proposed local joins can compute the similarity between each pair of different neighbors that a vertex v has. For large values of k, the cost of the local joins increase. Thus, performing these operations on samples of neighbors smoothes such a problem. The sampling is based on a sample rate ρ regarding the number of neighbor vertices that have not been visited yet. The early termination happens when the approximated *k-NN* can no longer be improved. The criteria defined by the authors is as follows: given the number of *k-NN* list updates (N) in each iteration, the algorithm finishes when this number becomes less than αkN, where α is a precision parameter that is roughly the fraction of true *k-NN* that are allowed to be missed due to early termination.

A recent proposal of a graph type for similarity searching is the *Navigable Small World* (*NSW*) graph. The *NSW* 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 [20]. 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 [54]. 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 graph's current state. The short-range edges tend to become long-range edges along the insertion process 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 [18, 9, 21]. However, the *k-NNG* and the *NSW* have being used as bases for the current state-of-the-art methods [55]. Therefore, we employed these base methods for the development of this work.

The following section aims at presenting a topic that estimates how complex the nearest neighbor problem is. We present different metrics and describe the most popular ones in the literature.

2.3 Dataset Metrics for Similarity Retrieval

A subject that has received attention in recent works is to analyze which properties make a dataset more challenging than others for similarity retrieval, including the approximate nearest neighbors (ANN) query, which we employ in this work. Researchers have explored properties such as the intrinsic dimensionality [29, 31, 33], the concentration phenomenon [56, 57, 34], and the relative contrast (RC) [35] to measure the complexity of datasets. This section presents an overview of these metrics and discusses how they relate to similarity retrieval.

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" [29]. 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 [30].

Works approaching the ANN retrieval commonly consider the intrinsic dimensionality of a dataset to measure its complexity. Korn et al. [29] show that intrinsic dimensionality is a key factor that affects the performance of nearest neighbor searches. In their experiments, they evaluate which datasets present the fractal property. Fractal is a general term used to describe both the geometry and the processes which exhibit self-similarity, scale invariance, and fractional dimension [58]. For instance, Figure 4 shows the Cantor ternary set. In the figure, each line refers to an iteration where the middle third is removed, which is the generation rule of this fractal. The fractal properties are clearly interpretable since all parts of this fractal have the same characteristics at different scales. Applying fractal properties to datasets, a set of points is considered a fractal if it has similar characteristics (e.g., geometric or statistical) at all scales. Real datasets are not fractals, but most of them present self-similar stochastic patterns. The authors estimate the intrinsic dimension through the fractal dimension, and their results show that the curse of dimensionality is not directly related to the embedded dimensionality but the fractal dimension.

We can divide the existing approaches for estimating the intrinsic dimension into two groups: the eigenvalue methods and the geometric methods. The former is based on global or local PCA (Principal Component Analysis), whereas the latter exploits the dataset's intrinsic geometry. For PCA methods, the intrinsic dimension is defined by the number of principal components that explain a certain amount of the variance [59]. On the other hand, geometric methods exploit the intrinsic geometry of the dataset and are usually based on distance concentration [57] or fractal dimensions [34].

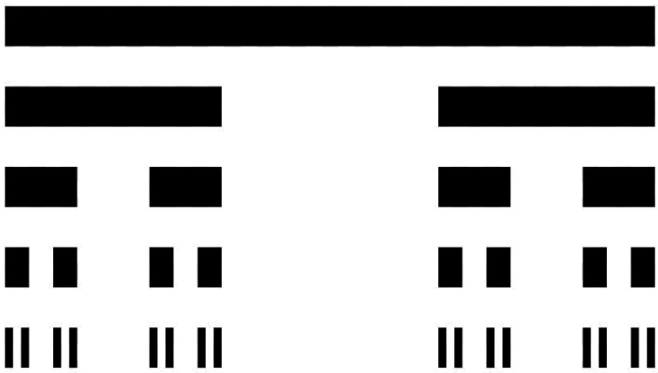


Figure 4 – The middle third Cantor set (adapted from [1]).

There are different methods to estimate the fractal dimension, such as the Hausdorff fractal dimension and the generalized fractal dimension [26]. A method [60] to estimate the correlation fractal dimension is as follows. Consider a set of points $X = \{x_1, ..., x_n\}$ in a metric space. Let the counting of pairs of points in X distant from each other up to a threshold r be given by Equation 2.3. The fractal correlation dimension is estimated as the slope of the plot of $\log(C_n(r))$ against $\log(r)$.

$$C_n(r) = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=i+1}^{N} 1\{||x_i - x_j|| < r\}. \tag{2.3}$$

Another classic approach to estimate the ID is the maximum likelihood estimator, proposed by Levina and Bickel [30]. Their definition is given as follows:

$$\hat{m}_k(x_i) = (\frac{1}{k-1} \sum_{j=1}^{k-1} \log \frac{T_k(x_i)}{T_j(x_i)})^{-1}, \tag{2.4}$$

$$\hat{m}_k = \frac{1}{N} \sum_{i=1}^{N} \hat{m}_k(x_i), \tag{2.5}$$

$$\hat{m} = \frac{1}{k_2 - k_1 + 1} \sum_{k=k_1}^{k_2} \hat{m}_k \tag{2.6}$$

where $T_k(x_i)$ denotes the distance from the point x_i to its k^{th} nearest neighbor, $\hat{m}_k$ is the estimated intrinsic dimension considering the k-nearest neighbors of each data point, and $\hat{m}$ is the final estimation of the intrinsic dimension of the dataset. In some cases, a dataset has different intrinsic dimensions at different scales. Therefore, k_1 and k_2 are input parameters that attempt to handle the issue by averaging the estimative over the range $[k_1, k_2]$.

Alternatively, the local intrinsic dimensionality (LID) has been employed in recent works [61, 31, 33] due to the flexibility of evaluating the intrinsic dimension in different regions. It can be estimated from Equation 2.4 (see [61]), and the provided vector enables us to extract different statistical metrics.

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. 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 [62, 63, 64]. To estimate how concentrated a dataset is, François et al. [34] proposed to measure the Relative Variance (RV) of the norm. The RV is defined in Equation 2.7:

$$RV_{\mathcal{F},p} = \frac{\sqrt{\mathrm{Var}(||X||_p))}}{\mathrm{E}(||X||_p)}, \tag{2.7}$$

where X is a d-dimensional random vector distributed according to the multivariate probability distribution function $\mathcal{F}$, and p defines the norm of the Minkowski family. The RV is a nonnegative rate, and small values indicate a high concentration of distances. Intuitively, it estimates the concentration by relating a measure of spread (variance) to a measure of location (expectation).

From the same perspective of estimating the complexity of a dataset, the Relative Contrast (RC) [35] is another measure employed to quantify how complex nearest neighbor searches are for a specific dataset. However, while the methods mentioned above lie on evaluating the distance distribution, the RC considers several data properties simultaneously, such as dimensionality, sparsity, cardinality, etc. Considering a dataset $X = \{x_i, ..., x_n\}$ a dataset containing n points and a query element q, where $x_i, q \in R^d$ are i.i.d. samples from a distribution $\mathcal{F}$, let $D^q_{min} = \min_{i=i,...,n} D(x_i, q)$ be the distance from q to the nearest database sample, and $D^q_{mean} = E_x[D(x, q)]$ the expected distance from q to a random database sample. The RC for the dataset X for a query q is defined as $C^q_r = D^q_{mean}/D^q_{min}$. The RC for the dataset X taking expectations with respect to queries is given as $C_r = E_q[D^q_{mean}]/E_q[D^q_{min}]$.

RC allows estimating the notion of the hardness of nearest neighbor searches in X. If C_r is close to 1, then, on average, the distance from a query element q to its nearest neighbor is almost the same as the distance from q to a random point in X. Thus, the smaller the C_r, the more complex the search. The RC can also be defined for the *k-NN*

setting as:

$$C_r^k = \frac{D_{mean}}{D_{knn}}, \tag{2.8}$$

where D_{knn} is the expected distance to the k^{th} nearest neighbor.

Overall, all the mentioned methods have the same purpose: to estimate how complex a dataset is for nearest neighbor searches. The hardness of the dataset is a valuable feature to define suitable parameters for indices for similarity retrieval. Therefore, we intend to employ these metrics as meta-features for modeling our meta-learning solution. We present more details about this in the next section, where we introduce basic concepts about meta-learning. First, let us summarize the described metrics in this section in Table 1.

Table 1 – Complexity metrics employed to measure the hardness of a dataset specifically for similarity search problems.

Property	Estimator	References
Intrinsic Dimensionality	MLE (ID and LID), PCA	[30, 61, 65]
Fractal correlation	Slope of the plot	[60]
Concentration of distances	RV, RC	[34, 35]

2.4 Meta-Learning

Machine learning (ML) has successfully been used to solve problems in different application domains. Consequently, growing demand for the optimization of ML algorithms has emerged. For many problems, it is hard to choose the ideal learning algorithm and its parameters as there usually are a lot of possible combinations. Some approaches in the literature treat algorithm selection and hyperparameter optimization as different tasks, and each of them is performed from scratch for every new scenario. However, recent alternatives have been directed towards automated machine learning (AutoML), including transfer learning [37] and meta-learning [66] techniques. AutoML consists of automatically choosing a suitable learning algorithm, feature preprocessing steps, and its hyperparameters [67, 68]. Transfer learning [37] builds a new model upon an existing model for a similar problem, taking advantage of the knowledge acquired by the existing model. *Meta-learning* addresses the idea of "learning to learn" by learning from a dataset of outcomes of a learning algorithm. This work proposes to employ meta-learning for recommending suitable configurations for graph-based methods for similarity searching.

Meta-learning differs from the traditional view of learning (a.k.a. base-learning) in the notion of what to learn [38]. 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. For instance, taking into account base-learners

(e.g., decision trees or neural networks), the learning is accomplished by accumulating experience from a single task. For base-learners, the more data, the better should be the predictive power. At a meta-level, the learning is achieved by accumulating experience from multiple applications. The more known problems, the more sophisticated the system will be to solve a new problem [37]. Thus, accumulating knowledge allows the meta-learner to induce predictive functions to find patterns among their underlying problems. In short, this approach learns how to learn across several domains.

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. *meta-attributes* or characterization measures): features to characterize datasets and/or problems.
- *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 [39]).
- *Meta-instances*: the junction of meta-features with their corresponding meta-targets.
- *Meta-dataset*: a set composed of meta-instances.
- *Meta-models*: learning methods that map meta-features to the meta-targets. That is, the predictors of the relative performance of the considered algorithms, e.g., the performance of a base-model, to induce a hypothesis for a given situation.

We can formally define meta-learning as follows. Consider a set 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}.

Adequately gathering the meta-knowledge is crucial for the development of a meta-learning application. We discuss this subject in the next section.

2.4.1 Gathering Meta-knowledge

Meta-features extraction, also known as dataset characterization, is one of the most critical steps of the meta-learning process [69]. Meta-features are data properties

that enable learning algorithms to find patterns among several domains. Thus, discovering and understanding how the dataset characteristics influence the algorithms' performance for different sets of parameters is essential to reach a suitable solution.

In the literature, there are several categories of meta-features. Classic meta-features describe the dataset using general, statistical, and information-theoretical measures [66]. These measures include the number of attributes, the number of instances, the degree of correlation between features, standard deviation, kurtosis, signal-to-noise ratio, average attribute entropy, etc. Classic meta-features are the most explored in literature [70, 71, 68].

Another alternative to represent meta-features is through a technique known as model-based characterization, which explores properties of models induced on the data. For instance, Bensusan et al. [72] presented an approach based on the use of a decision tree for dataset characterization. Attributes like depth, shape, and balance obtained from the tree are considered meta-features. Similarly, Leite and Brazdil [73] developed the learning-curve-based meta-features while Kalousis and Hilario [74] proposed the histogram-based meta-features. Moreover, the methods' performances can also be considered meta-features; this approach is known as the landmarking process [75].

Recently, new perspectives have also been explored. Measures based on the data complexity consider information like overlap in the values of an attribute, the presence of outliers, topological properties, etc. [76]. Complex networks, explored by Morais and Prati [77], consists of representing datasets through graphs. This approach assumes that each instance produces a node, and edges represent the relationship between these instances. The measures extracted by the authors from the graphs were the number of nodes, the number of edges, the graph's diameter, the clustering coefficient of each vertex, the standard deviation between pairs of vertices, and the ratio between the number of edges and the number of vertices in the graph.

Different meta-features are usually combined to improve the ability to characterize the dataset. The combination should consider the purpose of each of them, as their relevance depends on the problem domain. For instance, meta-features involving class information are irrelevant for meta-learning systems developed for unsupervised problems, which benefit from properties extracted from different clustering algorithms [42]. For time series domains, relevant features can be related to the slope of the regression models [40] while features based on the dataset's intrinsic dimensionality may be relevant for similarity retrieval.

The meta-targets vary according to the purpose of the meta-learning application. Meta-learning has been used for different purposes, such as algorithm selection [78, 43], prediction of algorithm performance [79], prediction of the training runtime [80, 81], decision if an algorithm should be tuned or not [41], recommendation of hyperparameters [82, 83], ranking algorithms [84], and so on. All these scenarios demand a high computa-

tional cost to gather the meta-targets to include in the meta-knowledge. For instance, an algorithm selection process firstly requires defining a set of models to be analyzed. Then, each model is applied to each dataset, where each of these applications generates a single meta-instance. Similarly, to recommend an optimal set of hyperparameters for a learning method, beyond the set of models, it is necessary to set a hyperparameter search space for each model and test every possible combination. Meta-targets can be directly represented by the algorithm performances or by a ranking considering the relative performances of the algorithms.

Finally, the meta-models are the product of the meta-learning process. Employing graph performances as meta-targets allows meta-models to predict how a configuration on a given dataset should perform. For instance, Yang et al. [85] used polynomial regressors for estimating the algorithm runtimes based only on the cardinality and dimensionality of datasets. Guerra et al. [86] train an SVM meta-regressor to predict the accuracy of classification algorithms under default parameters.

Although meta-models are able to provide suitable recommendations for unseen datasets, no tuning methods are considered. Often, such recommendations are used as a start point for an optimization procedure. This issue considers the fact that the meta-knowledge is gathered for a fixed parameter space. That is, given a meta-knowledge discretely composed of algorithm configurations, the meta-model will know only a fixed amount of configurations. To overcome this limitation, Kokiopoulou et al. [87] consider a parametrized softmax over all possible choices. The authors treat the neural architecture search problem, which consists of finding a proper architecture for a new dataset. In this case, having continuous parameter space allow the meta-model to maximize an estimated performance through simple gradient-based optimization.

In this chapter, we introduced basic concepts about similarity searches and meta-learning. Due to the recent success of graph-based methods, we present in the next chapter a deeper analysis of them and the difficulty of finding good algorithms and their parameter values. We also show how learning-based solutions have been employed to advance the state-of-the-art similarity search and other database areas.

3 THE PROBLEM OF PARAMETER RECOMMENDATION FOR GRAPH-BASED INDEXING METHODS

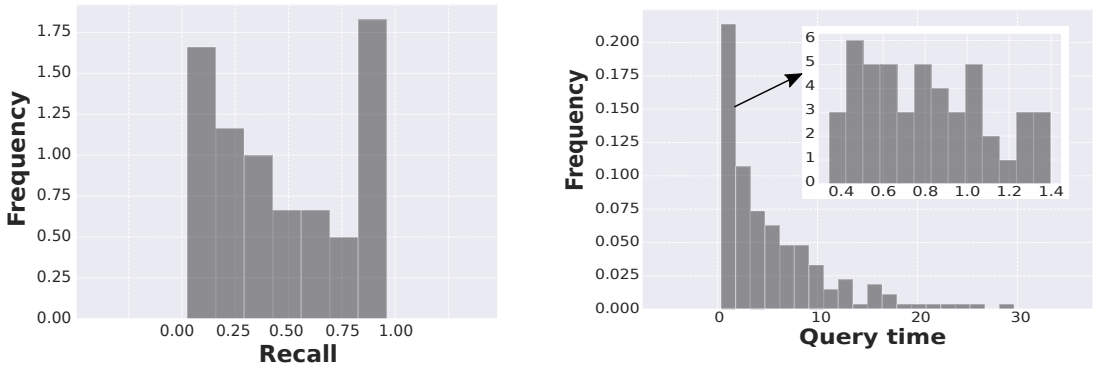
Currently, the literature results show that graph-based methods can present a better performance than the non-graph-based ones [52, 20, 25, 21, 88]. In addition, in our previous work [23] we demonstrated through several experiments that there is no better graph for all cases. That means that a graph can perform greatly for a specific dataset and poorly for other ones, although it does not mean that the graph cannot reach the desired performance for all cases. This happens because graph-based methods are very sensitive to their parameters, both for indexing and searching. Such a susceptibility brings up another drawback: finding an optimal algorithm configuration, which is usually accomplished by performing an exhaustive process. However, doing it for each new case becomes impracticable.

Our main goal is to develop an intelligent system capable of recommending a suitable proximity graph configuration for a given dataset. Firstly, this chapter shows works that provide comprehensive analyzes of graph-based methods for similarity searches and the impact of their parameters based on the experimental evaluation. Secondly, we discuss how learning-based solutions have been recently employed to advance the state-of-the-art in similarity searches and other database management areas. Lastly, we present works addressing the choice of algorithms and parameter values for similarity searches.

3.1 The Impact of Parameters for Graph-based Methods

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 parameter NN correctly is essential for an optimal trade-off, as it directly impacts construction time, memory usage, query quality, and execution query execution time. A low value for NN that results in high-quality queries is desirable since it leads to: (a) low construction time because of the search process in vertices' insertion; (b) low memory usage, as the fewer edges a graph has, the less memory it needs to be stored; and (c) low execution query time, as the lower the NN, the fewer adjacent vertices are evaluated in the search. However, low NN values lead to low-quality queries in most cases. Still, it is also possible to increase the quality of answers for low NN values, changing the parameters for the search algorithms. For instance, the *GNNS* algorithm depends on the R parameter. A low R parameter is desirable since it implies a low query time execution. However, low values for this parameter also lead to low-quality queries.

Subsequently, we present a detailed discussion about this issue. We show that



(a) (b)

Figure 5 – (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.

defining suitable values for these parameters has a major impact on the methods' effectiveness and performance and is a challenging problem. Also, 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. Our discussion considers typical scenarios.

The first scenario states that 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 this 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. Figure 5(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 Table 7). 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 random query elements. The recall rate of a query is the fraction of the actual 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 utterly 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. Figure 5(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*,

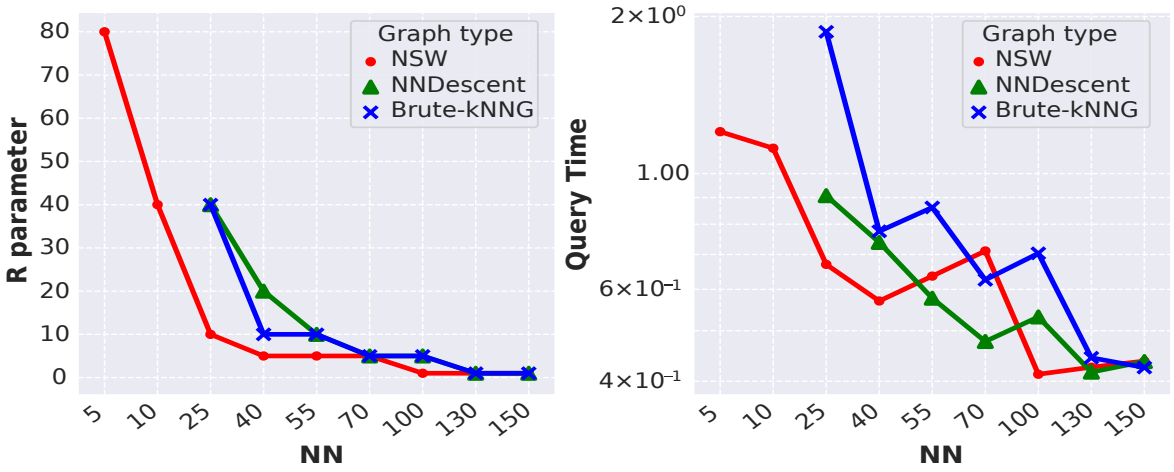


Figure 6 – The behavior of graph-based methods for the dataset Texture and increasing NN. a) Smallest number of restarts for the each graph and NN value. b) Query time for the corresponding configurations of the plot on the left.

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 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 can notice that almost two-thirds of the tested combinations of NN and R do not lie in the first bucket, which means that the 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 see that the variance regarding the average execution time is also large 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 a complete one as it requires choosing the best graph type and its configuration for a given dataset. Figure 6 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 that consumes less memory), 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 option. We can also see that choosing the graph type that is

the fastest in general, which is the *NSW* in this case, but with a poor configuration (e.g., $NN = 70$), may be the worst option among the graph types. 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, varying for different datasets and constraints. These reasons reinforce that the problem of recommending optimal parameters to configure graph-based methods is essential and challenging.

In the next section, we present some works of different areas that address the same difficulty of selecting suitable algorithms and/or parameter values. We focus on learning-based solutions due to the great success of these methods in the last years.

3.2 Parameter Recommendation for Database Systems

In the machine learning community, parameter recommendation is a common issue. Some of their solutions are based on intelligent systems to automatically select and configure learning algorithms according to a given problem domain [74, 79, 43, 83]. There are different methods to perform this task, and recently several proposals have lied in solutions based on meta-learning. Meta-learning has been employed in various problem domains, such as algorithm selection [43], parameter selection [83], algorithm performance estimation [84], deciding if an algorithm should be tuned or not [41], neural net architecture selection [87], pipeline recommendations [89], etc. The overall idea remains the same over these works, where the general objective is to map meta-features to the performance values achieved by each algorithm.

Similar approaches have also been applied to Database Management Systems (DBMS). Overall, most solutions make use of artificial intelligence techniques for developing learning-based database configuration, learning-based database optimization, learning-based database design, etc. Thus, learning-based solutions for DBMSs have become so popular in recent years that it has earned the nickname AI4DB [90].

As a remarkable example, Kraska et al. [91] presented the SageDB, an innovative perspective that addresses a new type of data processing system. The main idea is to take core database systems components, e.g., optimizer, index structures, etc., and build statistical and learning models for optimizing or replacing the existing approaches. The authors emphasize that this can be achieved by profoundly exploring data distribution. For instance, in [92], the same research group introduced the concept of learned indexes, claiming that index structures can be enhanced/represented/replaced by machine learning models. They presented a naive learned index, the RMI, whose idea is: (1) fit a simple learning model over the data; (2) use the prediction of this model to choose a subsequent model, which is more accurately fitted over a subset of the data; (3) repeat

the process until the last model makes the final prediction. Such a space partitioning procedure decreases the error narrowing the region to find the desired key. Although this initial approach presents some weaknesses, for instance, the tendency for overfitting, it largely outperformed a B-tree index in several situations.

Regarding parameter recommendation, DBMSs have a large set of parameters (knobs) that can be adjusted to improve their performance, such as the amount of memory for caching, how often data is written to storage, and transaction log buffer. This wide range of possibilities makes standard tuning approaches impracticable due to their high computational cost. Moreover, knobs are not universal nor independent, making their usability unique for each system. To overcome this issue, Aken et al. [93] combined supervised and unsupervised machine learning methods to implement a tool, called OtterTune, to recommend suitable configurations for new DBMS workload. The overall idea behind OtterTune is to build a repository of previous experiences, composed of workloads along with its knob configurations and its performances. This way, for a new DBMS target, the tool maps its characteristics to the existing workloads in the repository and provides a recommendation. The system can be divided into the following steps: (a) the workload characterization, which is performed measuring the DBMS metrics and clustering it through factor analysis and k-means methods; (b) identification and ranking of most crucial knobs, where a linear regression method (Lasso) is employed for feature selection; (c) mapping the similarity among workloads, which is performed measuring its Euclidean distances; For recommendation, each step above is performed for a new DBMS target, and the system recommends known configurations from similar workloads. The results show that recommendations are as good or better than ones generated by database administrators.

Similarly, Zhang et al. [94] and Li et al. [95] have also proposed a solution for knobs recommendations; however, both works used deep reinforcement learning (DRL). Their results have shown that DRL-based methods can significantly improve the tuning performance compared with traditional methods. Moreover, DRL presents several advantages, mainly for its ability to address the exploitation vs. exploration tradeoff. This behavior consists of concentrating efforts on exploiting a promising solution and also exploring similar alternatives.

Finding an optimal partitioning scheme is also a common problem in DBMS solutions, and usually, this is performed by the database administrator. This procedure is a non-trivial task as it has a significant impact on the overall performance. Thus, Hilprecht et al. [96] proposed a learned partitioning advisor also based on DRL to overcome this challenge. Their goal is inducing a learner capable of suggesting a partitioning scheme for an observed workload. A DRL agent is employed to learn the tradeoffs among it by deploying different partitioning and observing query runtimes. The system's overall idea

is to measure how much overhead is needed to answer a query given a particular partitioning. This simulation enables the system to estimate the query costs considering the partitioning and its attributes (scheme and table sizes). From such estimates, the agent can learn the tradeoffs and finally suggest partitioning schemes.

The works mentioned above are more generic for our purpose; that is, they treat the automated configuration/selection of algorithms in different database areas. The following section focuses on how the similarity search community addresses these challenges.

3.3 Parameter Recommendation for Graph-based Methods

Although algorithm and parameter recommendation, 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 indices based either on the user intuition or exhaustive evaluations. These approaches lead to suboptimal configurations and/or are excessively time-consuming since the identification of 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. [88] on state-of-the-art Approximate Nearest Neighbor (ANN) algorithms to provide a better understanding of their general behavior. The authors evaluated 16 algorithms and analyzed their advantages and disadvantages on 20 datasets. This comprehensive experimental study showed the strength and weaknesses of the evaluated methods according to each dataset's characteristics. Also, the authors presented some concepts for understanding why some datasets are harder than others. Their analysis showed how existent metrics, such as Relative Contrast [35] and Local Intrinsic Dimensionality [61], can characterize datasets providing valuable insights for choosing ANN algorithms and their parameters.

Similarly, the *ANN-Benchmark* was proposed to standardize the evaluation of ANN algorithms [25]. This benchmark enables comparing a wide range of ANN algorithms and their configurations on several real datasets. The system was developed to evaluate tree-based, hash-based, and graph-based methods, considering several parameters. Still, the authors state the need for developing new tuning methods, which requires getting a better understanding of the impact of the dataset's intrinsic properties 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 [23]. We implemented six base graph types and three query algorithms in a common platform, being one of them for exact retrieval for data in metric spaces. We performed an extensive evaluation on real datasets showing relevant trade-offs in the behavior of the main base graph types

according to construction and query parameters. Synthetic datasets were also employed for a better understanding of parametrization according to the dataset properties. We varied different properties to generate synthetic datasets, such as dimensionality, number of clusters, and distribution. The results allowed us to identify general patterns, such as increasing the *NN* value rises the recall, and reduces the query time up to a limit when it starts to increase due to the cost of vertex expansion. Another pattern is that increasing the number of restarts in a query improves the recall up to a limit at the cost of demanding more time to execute the query. After that limit, the execution time increases with no benefit to the quality of the result. However, finding the ideal trade-off between such parameters for different datasets is challenging, as it depends on the dataset properties and user requirements.

To the best of our knowledge, the only method of auto-selecting a suitable algorithm configuration for similarity searches was proposed by Muja and Lowe [45]. Their purpose is to minimize a cost function based on query time, indexing time, and memory usage. The procedure is performed in two steps: first, a grid search strategy is used to find the parameter values that minimize the function; subsequently, a local exploration is realized to fine-tune the parameters obtained. Although the user can provide only a subset to save time, the grid search step, in general, is still an expensive approach. A limitation of this work is the fact it only accepts tree-based methods.

A related work that applies a learning-based method for refining proximity graphs was introduced by Baranchuk and Babenko [97]. Knowing that graph-based methods have outperformed other ANN algorithms, the authors state that new graph-based proposals are always evaluated over a limited number of datasets, which does not guarantee a universal application for them. Also, in most cases, these methods are based on a new construction heuristic, not an optimization of an existing one. Therefore, they present a new construction method for proximity graphs that explicitly increases the search efficiency by exploring the probabilities of the edges of a constructed graph. The overall idea is to employ a reinforcement learning agent that, given a constructed graph, performs several queries over the graph, dropping edges that are never used. This approach decreases the average vertex out-degree significantly. Moreover, the results presented show that these new methods achieve higher recall rates under the same number of distance computations. The major contribution is that their method can be used to refine any state-of-the-art proximity graph. However, this work does not solve the problem of recommending parameters for proximity graphs as it depends on a pre-built graph, which requires an initial parameter setting. Nevertheless, if the initial setting yields insufficient edges, the refinement will be limited. Similarly, if the initial setting produces too many edges, the construction cost will be unnecessarily high, with a large number of edges being dropped during the refinement.

Another related subject is to apply learning techniques for improving search algorithms in ANN problems. Li et al. [98] proposed an intelligent approach that learns when a query should stop or not, achieving the same accuracy as current state-of-the-art methods but with less latency. Overall, ANN methods use the same termination condition for all queries, which is usually a predefined parameter. The authors stated that fixing such conditions is a considerable limitation, and it leads to high latency. The problem is that some queries are more complex than others; thus the easiest ones could terminate before the harder ones. To accomplish this goal, firstly, the output for training is generated by performing actual ANN searches to know how much effort is needed for each query to find the true nearest neighbor. As input, the model receives the query (the feature vector itself), its distance to the initial object of the index structure, and the query's current state after a certain amount of search. An ML model is induced according to this collected data. Given a new query, the model returns a value to indicate the minimum number of distance evaluations needed to find its nearest neighbors. Although this approach is useful to speed up retrieval on graph-based methods, it is orthogonal to the problem of recommending their parameters.

Selecting suitable algorithms and parameter values for performing similarity searches is not a trivial task. However, there is a lack of works investing in learning-based solutions to address this difficulty in the ANN community. Therefore, in this work, we aim at developing an intelligent system based on meta-learning techniques to recommend graph-based methods along with its parameters. We present more details about our proposal in the next chapter.

4 A META-LEARNING APPROACH FOR AUTO-CONFIGURATION OF PROXIMITY GRAPHS

The main objective of this master's thesis 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 chapter presents our proposal for recommending proximity graphs and the main aspects adopted through this thesis. Initially, we formally describe the problem recommending proximity graphs by employing meta-learning techniques (section 4.1). Section 4.2 shows an overview of the proposed framework. Subsequently, we detail the techniques employed in the main tasks of our framework. Section 4.3 shows our strategy for gathering meta-knowledge, detailing the dataset descriptors and performance measures extracted from the graphs. Finally, section 4.4 discuss how we propose to define the meta-models to predict the expected retrieval performance, and provide the recommendation according to the chosen optimization goal.

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 as follows. Let D be a set of complex data datasets. Let $d \in D$ be a dataset and θ 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.

Let the construction of an index on a dataset d or the batches of queries executed on this index, using a parameterization θ, result in a performance value $p_{d,\theta}$. Let d_{mf} be a vector of meta-features describing a dataset d. Let $\mathcal{D}$ be a meta-dataset composed of meta-instances with the form $I = (X, y)$, where $X = \langle d_f, \theta \rangle$ is a pair containing a vector of meta-features and a graph configuration; and $y = p_{d,\theta}$ is the corresponding meta-target. Finally, let $\hat{f}_M(X) = \hat{y}$ be a function learned by a meta-model M trained on a meta-dataset $\mathcal{D}$, 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 can be used to minimize the query time or the amount of memory demanded by the method by minimizing 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 $X_i = \langle d_{f_{new}}, \theta_j \rangle$, where $\theta_j \in \Theta_C$. Finally, we select the parameterization θ_j whose prediction $\hat{y}$ satisfies the optimization goal ψ and return θ_j as the recommended parameterization for the case.

4.2 A Framework to Provide Recommendations

To develop the solution following the proposed problem modeling, we have three major 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 and provide 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.

Figure 7 summarizes the architecture of the proposed meta-learning-based recommendation system. It illustrates the steps of gathering meta-knowledge, performing the meta-learning, and getting the recommendations according to the user's inputs. The task of building the meta-dataset takes as input a set of datasets, a set of graph-based methods, and a set of parameter values. The set of graph configurations are combinations of graph type and values for the considered parameters. For each dataset in the set of datasets, the architecture generates meta-instances composed of meta-features extracted from the dataset (characterization) and a graph configuration. The meta-dataset is composed of meta-instances that associate the meta-features with meta-targets. A meta-target is a performance measurement (e.g., average query time or recall) obtained by running batches of queries using the graph-based method built for the dataset using the meta-instance's construction and query configuration values (performance measurement).

Subsequently, the recommender applies a meta-learner to build a meta-model base, which can be composed by different regressors trained using the meta-dataset. To obtain

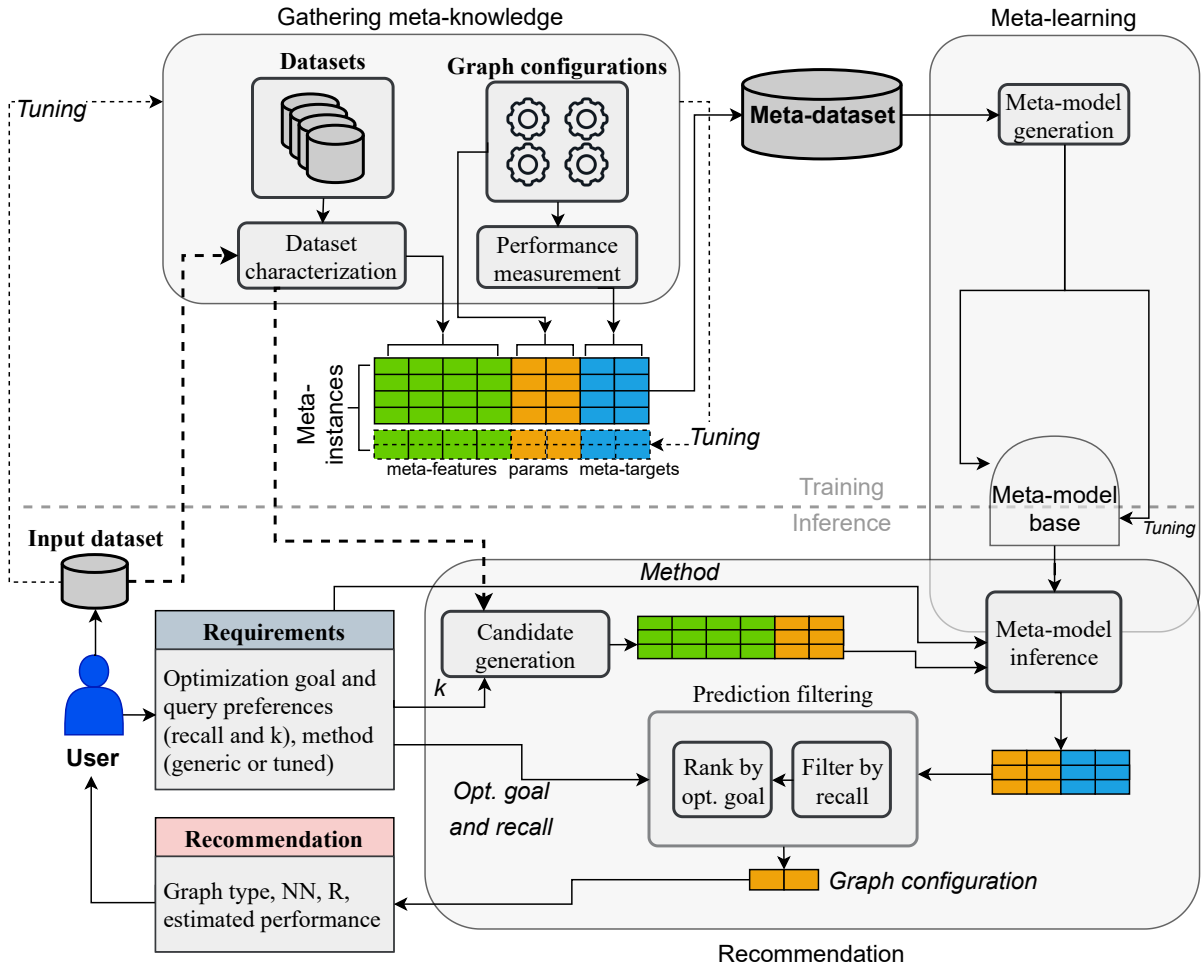


Figure 7 – The proposed framework, illustrating how the overall process of building the meta-dataset, instantiating and using the meta-model, and generating recommendations according to inputs provided by a user.

a recommendation, a user provides a new dataset and some preferences, including the requirements to be satisfied (e.g., query preferences, specific method) and the optimization goal (e.g., query time or memory usage). The input dataset is characterized, and the recommender generates candidates by defining a set of meta-instances simulating different parametrizations for the input dataset and infers the corresponding performance measures (meta-targets). Finally, it filters the meta-instances according to the performance measure and optimization goal and returns the user the best parametrization. If there are two or more parametrizations tied in the top position, all of them are returned.

Figure 7 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 not only for characterization but also for performance measuring on 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 performance measurement should be effective for other hardware as our recommender is based on relative performances using ranking. Moreover, in Chapter 5 and Chapter 6 we present two different strategies for generating meta-models, showing that the framework is valid for a range of meta-learning alternatives.

The following sections detail the techniques we propose for gathering meta-knowledge, building meta-models, and generating recommendations.

4.3 Strategy for Gathering Meta-Knowledge

This section details the proposed strategy to gather meta-knowledge. The construction of the meta-dataset is divided into three steps: (a) defining a set of datasets and extracting meta-features from each of them, (b) establishing a set of graph-based method types and parameter values, and (c) extracting meta-targets by measuring the performances of every graph-based method configuration on every dataset.

There are a plethora of datasets of complex data available. The first decision is to select a reduced set 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, we have as a starting point 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 should improve the generalization of our approach.

After collecting datasets and defining the indexing methods to support them, building the meta-base comprises the following steps. First, we perform the characterization of each dataset extracting its meta-features. Subsequently, we run batches of experiments to extract the meta-targets regarding each graph configuration. 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. The following subsections define the meta-features and meta-targets we employ in our solution.

4.3.1 Meta-features

As mentioned in section 2.4, 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 indices for complex data, here we outline meaningful metrics to employ as meta-features in our proposal.

We propose to employ all the metrics described in section 2.3 as meta-features, as they aim at measuring the complexity of datasets regarding similarity retrieval, including the (approximate) nearest neighbor search problem. Although there are different estimators available in the literature for intrinsic dimensionality, very 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, it is feasible to employ different methods in order to increase the meta-dataset.

We also propose to include classical meta-features that describe general, statistical, and information-theoretical measures described in subsection 2.4.1 to compose the meta-dataset. Many general measures, such as cardinality and embedded dimensionality, have been widely explored by the similarity search community to evaluate indexing structures and searching algorithms.

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. We summarize the meta-features composing the meta-dataset built in this work in Table 2.

4.3.2 Meta-targets

The evaluation of indexing structures is given by analyzing performance measurements simultaneously. Regarding graph-based methods, a few measuments are conflicting, such as the retrieval speed and the retrieval quality of approximate nearest neighbor queries. In this case, the challenge is to find the ideal trade-off between them, as increasing the retrieval quality usually also leads to increasing the execution time. Besides, more edges in the graph lead to more retrieval quality for a single search. However, adding edges increases only the query execution time with no additional benefit to the quality after a stagnation point. Nevertheless, a large number of edges leads to a proportionally expensive index construction and large memory consumption.

The meta-targets should cover a variety of measures that impact similarity retrieval using graph-based methods. 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 having to build an index and/or run 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;

Table 2 – All attributes of the final 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 dimesnionalities.
	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.

- 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 3.1, 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. A usual query tuning parameter for these graph-based methods is the number of traversals starting from distinct vertices to increase the result accuracy. For example, the *GNNS* is a well-known k-NN algorithm that relies on such a tunable number of traversals — the 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 the structure according to the given search parameter. 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. The amount of memory to store a graph structure is also an important metric to consider. However, this measure is directly related to the index parameter NN (the number of neighbors that each vertex should be connected to). It is known that a high NN value implies a large number of edges, which means high memory usage.

Our framework considers that the meta-dataset grows over time, including new datasets, either by running offline performance measurements or online measurements 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 loss of eventually adding noise. The graph configuration attributes and meta-targets complete the meta-dataset our proposal uses, as shown in Table 2.

4.4 Meta-learning and Recommendation

After gathering all meta-knowledge for building the meta-dataset, we can induce meta-models to estimate the relative performances of graph-based methods to provide recommendations, as follows.

4.4.1 Definition of Meta-Models

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

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 (*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 have knowledge of a dataset similar to the input one.

We present two meta-learning strategies for our framework in the following two chapters. 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.

4.4.2 Recommendation

As described in section 4.2, the user receives recommendations by entering with 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 performs the inference of 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 3. 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 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.

Table 3 – 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

5 A GLOBAL META-LEARNING RECOMMENDER INSTANTIATION

This chapter introduces a global strategy to instantiate our recommendation framework's meta-learning building block, published in [44]. We start by providing a general overview of the instantiation in the next section, followed by the experimental setup in section 5.2. Subsequently, in section 5.3 we discuss the achieved results, including the relative importance of the meta-features (subsection 5.3.1), the accuracy of our proposal (subsection 5.3.2) and its effectiveness compared to a grid search procedure (subsection 5.3.3).

5.1 Overview of the Instantiation

Following the proposed framework in Figure 7, here we present our strategy to instantiate a meta-model. We approach the problem of predicting different performance measurements as a multi-output regression problem. This strategy consists of fitting one regressor for each meta-target mt_i, as illustrated in Figure 8. We consider it a global instantiation because we induce the meta-models from the entire meta-dataset.

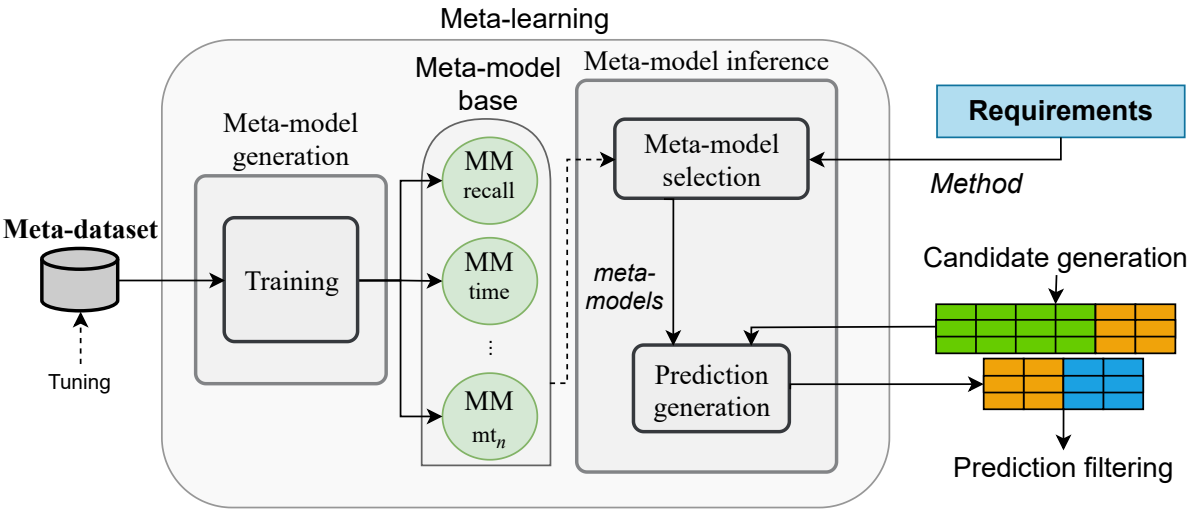


Figure 8 – A global instantiation of meta-models following the framework in Figure 7. There is a meta-model for each of the n meta-targets.

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. The following sections present the experimental setup and results achieved evaluating the instantiation, showing that it is simple but effective in many cases.

5.2 Experimental setup

This section details the setup to generate and evaluate the global meta-learning instantiation, including the datasets employed, the dataset characterization, performance measurement, and the learning method.

5.2.1 Datasets

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 respective 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; 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 cardinality, dimensionality, number of clusters, and each cluster's distribution standard deviation. The dataset generation was performed using the Python library Scikit-learn[1]. Table 4 summarizes all the values of the manipulated properties from synthetic datasets and an overview of the real datasets.

5.2.2 Dataset Characterization

The characterization generates the meta-features describing the datasets. Most of the meta-features employed were based on general, statistical, and information-theoretical measures. These measures are extracted from each column of a dataset. Thus, each of them

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

Table 4 – The real and synthetic datasets employed and their characteristics.

	Title	Size	Dimensions
Real	*Moments*	68,040	9
	Texture	68,040	16
	Histogram	68,040	32
	MNIST	70,000	784
	SIFT	1,000,000	128
	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}	

generates two meta-features, represented by its mean and standard deviation. We used the tool PyMFE [99] to extract these features.

Regarding, the complexity metrics w.r.t. the similarity search problem [100, 101, 33], described in Table 2, we included only the intrinsic dimensionality (ID) in this chapter. This metric was estimated by averaging of the local intrinsic dimensionalities using the Maximum Likelihood Estimation proposed by Amsaleg et al. [61].

5.2.3 Performance Measurement

The graph-based methods selected in this work are the *Brute-kNNG*, *NN-Descent*, and *NSW*, using the *GNNS* search algorithm. We used implementations in the C++ library NMSLib (Non-Metric Space Library) [102]. 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 [23]. 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*.

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. It must be notice that, although in Chapter 4 we suggested using more meta-targets, this chapter's instantiation only employed the recall and query time.

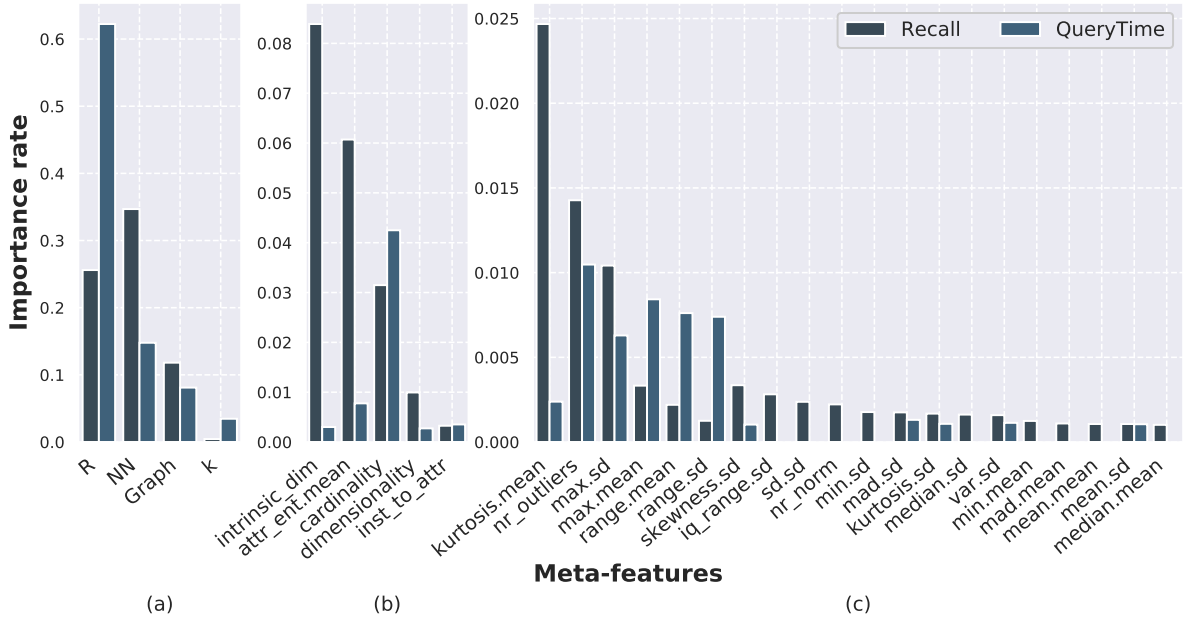


Figure 9 – The importance rate of each meta-feature per target (recall or query time) and category: (a) graph configurations, (b) general and information-theoretical, and (c) statistical.

5.2.4 Learning Method

We used the Random Forest (RF) method using default hyperparameters from Scikit-learn to induce the meta-models, and a 5-fold Cross-Validation strategy to validate them. We selected the RF for its excellent prediction performance and fast training and inference, reported in several recent works [103, 104], simple parameterization [105], and capacity to evaluate feature importance. The evaluation of such importance helps to understand the impact of each meta-feature for each specific meta-target.

5.3 Experimental Results

In this section, we present experimental results regarding our first evaluation, published in [44], presenting a discussion about the relative importance of the meta-features, the prediction accuracy of the meta-models, and the effectiveness of our instantiation's recommendations compared to a grid search procedure.

5.3.1 Analysis of Meta-feature Importance

In this section, we present an analysis of the importance rate of each meta-feature to predict the meta-target. These rates were measured by the meta-models performing 5-fold cross-validation over the meta-dataset. Our results are presented in Figure 9. Overall, for both meta-targets evaluated, the most relevant meta-features were the construction parameter NN, the search parameters R and k and, the graph type. In the meta-models,

all the following splits depend on the graph type and its parameters. Nonetheless, other meta-features also contribute to the prediction, refining, and determining the proximity graph's behavior on a given set of dataset descriptors. However, there is also some intuitive analysis that can be taken into consideration. Regarding recall, we can observe that the most relevant meta-feature is the construction parameter NN. The relevance of the NN is implied by the fact that the more edges a graph has, the better its query recall rate is. Similarly, for query time, we have the query parameter R as the most important meta-feature as the higher the number of restarts R is, the longer the query execution time is, and vice-versa.

Another intuitive interpretation of the query time is that the cardinality, which also has a high importance rate, represents the number of nodes in a graph. Naturally, the more nodes a graph has, the higher is the time spent in the search. Moreover, the high importance rate of the ID measure for recall analysis corroborates previous research. Excluding the graph configuration meta-features, the ID had the highest rate, while the embedding dimensionality showed no relevance in this case. This result reinforces the claim stated by [29], which says that it is the intrinsic dimension that determines the search performance.

5.3.2 Prediction Accuracy of the Meta-models

In this subsection, we present our results on the prediction accuracy of the meta-models regarding the real datasets used in this work. The synthetic datasets were added to the meta-database to provide a wider diversity of dataset characteristics. We evaluated our approach by using the three different strategies described in subsection 4.4.1, as follows.

- Generic Meta-Model (*GMM*) – uses for training all meta-instances of our meta-dataset regarding all datasets, except the goal dataset;
- Tuned Meta-Model using Grid Search (*TMM-GS*) – uses for training all meta-instances of our meta-dataset regarding all datasets, except the goal dataset, plus tuning meta-instances generated by a grid search performed on the goal dataset; and
- Tuned Meta-Model using Subsets (*TMM-S*) – uses for training meta-instances of our meta-dataset regarding all datasets, except the goal dataset, plus tuning meta-instances regarding subsets of the goal dataset.

The GMM strategy simulates generating a recommendation for an unseen input dataset. The TMM-GS simulates a fine-tuning of the meta-models by increasing the meta-dataset with meta-instances generated by a grid search with a limited parameter space.

Table 5 – Relative performances of the generic and tuned meta-models.

Goal Dataset	GMM				TMM-GS				TMM-S			
	Recall		QueryTime		Recall		QueryTime		Recall		QueryTime	
	r^2	RMSE	r^2	RMSE	r^2	RMSE	r^2	RMSE	r^2	RMSE	r^2	RMSE
Histogram	0.350	0.135	0.980	0.249	0.605	0.130	0.961	0.338	0.996	0.012	0.998	0.068
MNIST	0.765	0.111	0.694	1.097	0.617	0.173	0.920	0.559	0.997	0.014	0.998	0.068
Moments	0.955	0.034	0.989	0.179	0.973	0.031	0.979	0.241	0.991	0.019	0.998	0.065
SIFT	0.807	0.132	0.932	0.524	0.568	0.247	0.803	0.932	0.983	0.049	0.984	0.260
Texture	0.978	0.024	0.962	0.344	0.990	0.022	0.951	0.378	0.996	0.012	0.998	0.058

Lastly, the TMM-S simulates a scenario in which the meta-model already knows datasets with similar properties to the input dataset.

Table 5 presents the relative performance of the induced meta-models considering two evaluation metrics: the Coefficient of Determination (r^2), which estimates how much variance of a dependent variable can be explained by the independent variable, and the Root Mean Squared Error (*RMSE*), which measures the differences between predicted and true values. The higher the former one and the lower the latter one, the best is the performance. From the results, we can observe that the generic meta-models (GMM) strategy reached good scores for query time and fair scores for recall for most of the datasets. The higher accuracy of query time compared to recall is because meta-features were more supplementary for query time than recall. For TMM-GS, it was expected a slight improvement compared to the GMM. On the other hand, the most tuned meta-model TMM-S achieved high scores for both recall and query time. Therefore, by investing effort to generate meta-instances from the goal dataset, the user can achieve a superior recommendation.

5.3.3 Recommendation Effectiviness

In this section, we discuss the recommendation effectiviness provided by our approaches compared to a Grid Search (GS).

While we assume that the meta-dataset used to induce the meta-models may be continuously enhanced with new meta-instances, the grid search usually limits the number of parameter value combinations of a method to a reduced amount due to the cost of testing all of the combinations. Therefore, it is fair to consider that the number of combinations in the meta-data set is larger than the number of combinations tested in a grid search (GS) for a specific dataset.

Under this assumption, we emulated a grid search using the subset of entries in the meta-dataset such that $NN = \{10, 25, 70, 150\}$ and $R = \{1, 10, 40, 120\}$. These were the same parameters used to evaluate the *TMM-GS*. In this analysis, we set the constraint of achieving a minimum average recall of 0.90 and evaluated two optimization criteria: the shortest query time, in Figure 10, and the lowest memory usage, in Figure 11. In the first figure, the recommendation is performed by choosing the lowest predicted query time; however, we plot the actual query time to see how far it is from the optimal. The optimal

refers to the best meta-instance according to each case. For subsets, the TMM-GS and TMM-S were not employed. Missing bars in figures mean wrong recommendations, i.e., when the method fails to satisfy the recall constraint due to a poor prediction.

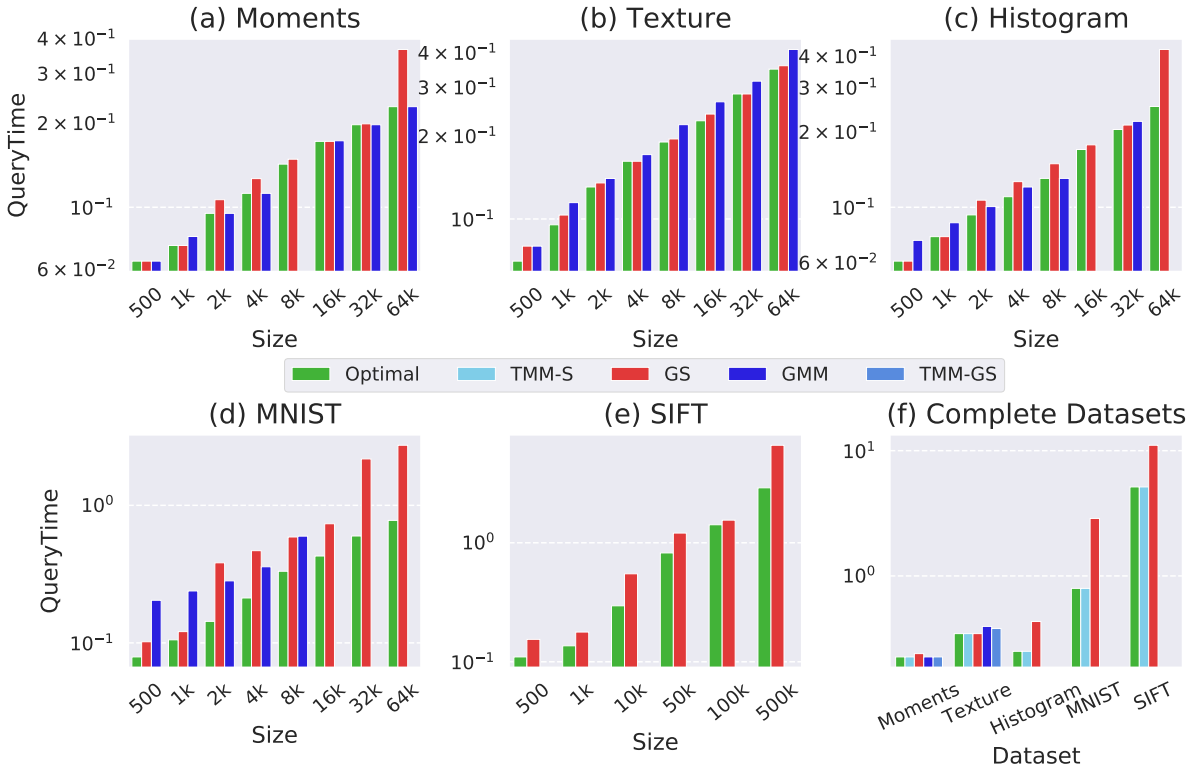


Figure 10 – Comparison of the recommendations provided by the methods attending to the requirement of the lowest query time.

The plots in Figure 10(a-e) show recommendations for subsets of each dataset while Figure 10(f) shows recommendations for the complete datasets. For subsets of Moments and Histogram, the GMM overcame the GS in most cases, while, for complete datasets, the TMM-GS achieved the optimal for moments, but for Histogram, it provided wrong recommendations. For Texture, although the GMM and TMM-GS did not overcome GS in any case, the average difference of query time between both was around 11%. Overall, although the GMM did not overcome the GS in all cases, both are frequently very close, which is positive considering that the computational cost of a GS procedure is substantially higher than using the GMM. For MNIST and SIFT, the GMM performed very poorly, providing wrong predictions in most subsets. Likewise, for complete datasets, neither GMM nor TMM-GS provided correct recommendations. The TMM-S reached the optimal in all cases.

Following a similar layout, the plots in Figure 11(a-e) show recommendations for subsets while Figure 11(f) for the complete datasets. Every time our methods generated recommendations satisfying the recall constraint, they reached the optimal, overcoming the GS in these cases. However, in many cases, our methods provided wrong recommen-

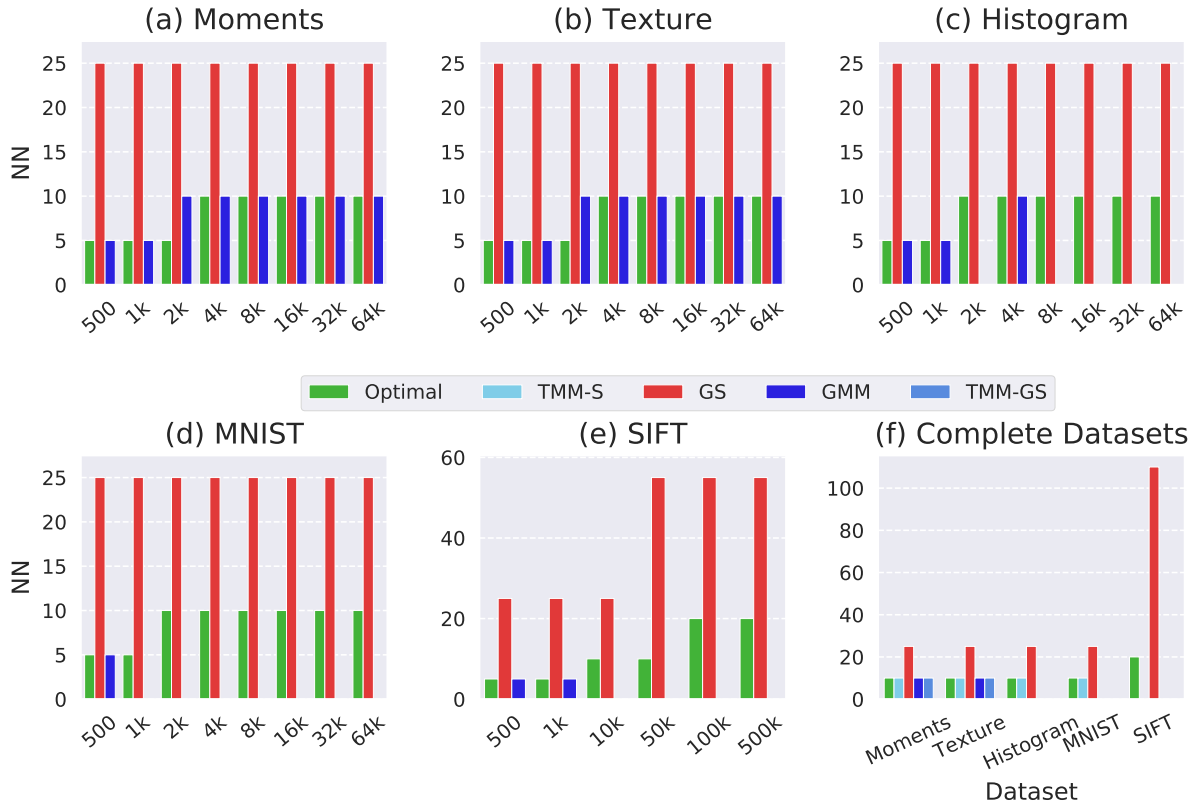


Figure 11 – Comparison of the recommendations provided by the methods attending to the requirement of least memory usage.

dations. It can be seen that the GMM excelled for *Moments* and *Texture* but, for the remaining datasets, it only provided correct recommendations for small subsets. Analyzing the complete datasets, the TMM-GS tied with the GMM while the TMM-S was the best strategy, reaching the optimal for all datasets but *SIFT*. These results reveal that the recall was frequently overestimated for complex datasets such as *MNIST* and *SIFT* due to their high dimensionality.

To better understand the recall prediction accuracy throughout the datasets, Figure 12 shows the actual recall rates paired with their corresponding predictions split into intervals. Columns refer to the methods, and rows refer to the datasets.

For the Texture dataset, we can notice a slight improvement at each meta-model. GMM provided a few poor predictions for recall rates in the interval [0.5, 0.7), TMM-GS smoothed such errors, and TMM-S achieved excellent predictions for all intervals. Similarly, for Moments there were slight improvements from GMM to TMM-GS; however, TMM-GS could not overcome TMM-S, which also performed very well. Although GMM and TMM-GS did not achieve great performances as TMM-S, both performed reasonably well in general. For Histogram and MNIST, we can notice a similar behavior of slight improvements from GMM to TMM-GS; nevertheless, both performed poorly in general. Again, TMM-S overcome all methods providing great predictions. Lastly, GMM

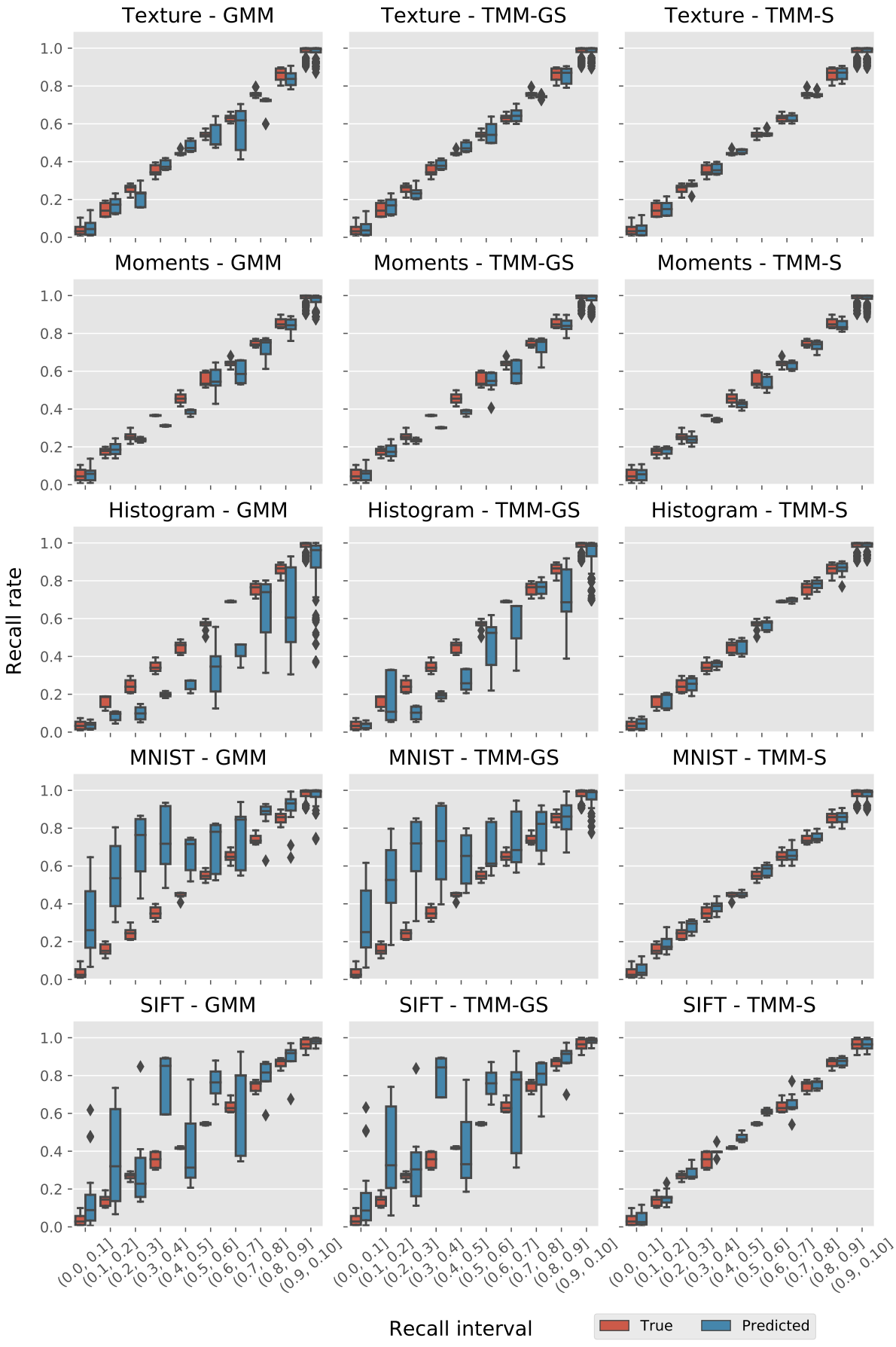


Figure 12 – Accuracy of the predictions of the meta-models per recall interval for each dataset.

and TMM-GS performed likewise for SIFT dataset, while TMM-S achieved good results.

Notice that in most cases, all meta-models provide high-quality predictions for recall rates greater than 0.9. Thus, it can be taken into account that users will frequently require solutions that achieve high performances in ANN problems. Therefore, such an interval of accurate predictions is a positive point for a recommendation system because it shows high efficacy around the interval of most interest. However, we can conclude that although predictions involving high recalls are accurate, the poor predictions of other intervals are directly interfering with the recommendations. For instance, in Figure 12, see GMM predictions for the datasets MNIST, in interval [0.3, 0.4), and SIFT, in interval [0.8, 0.9). The intervals refer to actual recall rates reached by certain configurations; however, some predictions regarding these recall rates were greater than 0.9. Thus, our instantiation can infer a high recall rate for a given graph configuration that actually performs poorly.

Table 6 – Recommendations by GMM that satisfy the requirements of $recall \geq 0.90$ with the lowest *query time*.

Dataset	**True Recall**	**Predicted Recall**	**Graph**	**NN**	**R**
Texture	0.898	0.903	NND	40	5
Moments	0.926	0.914	NND	100	1
Histogram	0.906	0.924	NND	100	1
MNIST	0.782	0.923	NND	70	5
SIFT	0.876	0.904	NSW	110	10

Table 6 presents an example to illustrate this limitation, showing graph configurations recommended by the GMM. In this case, the requirement was the fastest graph configuration that achieved a minimum recall of 0.9. Although the recommendation also depends on the query time predictions, we are not showing them here to emphasize only the problem regarding the recall. The table clarifies the pointed problem, especially for the datasets MNIST and SIFT. Notice also that for Texture, the prediction was the closest to the actual recall. However, we consider it is a wrong recommendation because the actual recall does not satisfy the posed requirement. If such an error was acceptable the recommendations not shown in the plots of Figure 10 and Figure 11 could be helpful to the user.

6 A DATASET-SIMILARITY-BASED META-LEARNING RECOMMENDER INSTANTIATION

In the previous chapter, we presented a global meta-learning instantiation. The results showed that the overall accuracy is good; however, the global instantiation produces some poor recommendations. This chapter presents an improved strategy that significantly increases the meta-target prediction quality, yielding superior recommendations.

The strategy approaches the steps of building the meta-dataset and managing the meta-models. We enhanced the meta-dataset by employing new meta-features for characterizing datasets. We also added new image feature datasets to assess the impact of our framework's evolution by enriching the meta-database. Furthermore, we included in the framework a data augmentation step using an interpolation procedure. The new meta-features and the data augmentation boosted the meta-models' prediction accuracy.

Regarding managing the meta-models, we moved from global meta-models to dataset-similarity-based meta-models, i.e., meta-models for groups of datasets with similar properties. We embed the datasets into a similarity space defined by the meta-features most impacting the meta-target for each meta-target. Then, we cluster the datasets in each similarity space and induce meta-models for each cluster. The recommendation for a new input dataset relies on predictions provided by the meta-models from the cluster closest to the input dataset. Such a procedure "overfits" the cluster meta-models without compromising the generalization since embedding the datasets in the similarity space allows us to find the meta-models best-suited for the input dataset.

We start this chapter by providing details of this new strategy to implement our recommendation framework. Section 6.1 describes the proposed dataset-similarity-based instantiation to manage meta-models. Section 6.2 presents the new meta-features and the data augmentation method. Subsequently, we present an experimental evaluation of the strategy. Section 6.3 shows results that support the decisions we made building the instantiation. Lastly, section 6.4 presents an evaluation of methods following the dataset-similarity-based instantiation compared to baseline methods and methods following the global instantiation. The results are shown, classifying the methods into two classes: quick and tuned methods. The former one regards recommendation methods that do not require any extra time to recommend. The latter one concerns methods that require an investment of time to receive recommendations.

6.1 Instantiation Overview

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:

(a) evaluate which meta-features are the most relevant for a given meta-target;

(b) create a dataset space according to these meta-features and apply a clustering algorithm;

(c) induce meta-models tailored to the datasets from each cluster;

(d) 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 Figure 13. Initially, we perform a feature selection step for each meta-target using the whole meta-dataset enhanced with new complexity meta-features and data augmentation, as described later in section 6.2. 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. For each cluster, the strategy induces a meta-model for each meta-target and adds it to the meta-model base.

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 in the previous chapter, respectively, 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 present results in this chapter showing that methods using our dataset-similarity-based strategy outperform both the baselines and the methods following the global strategy. The following subsections detail the main instantiation components.

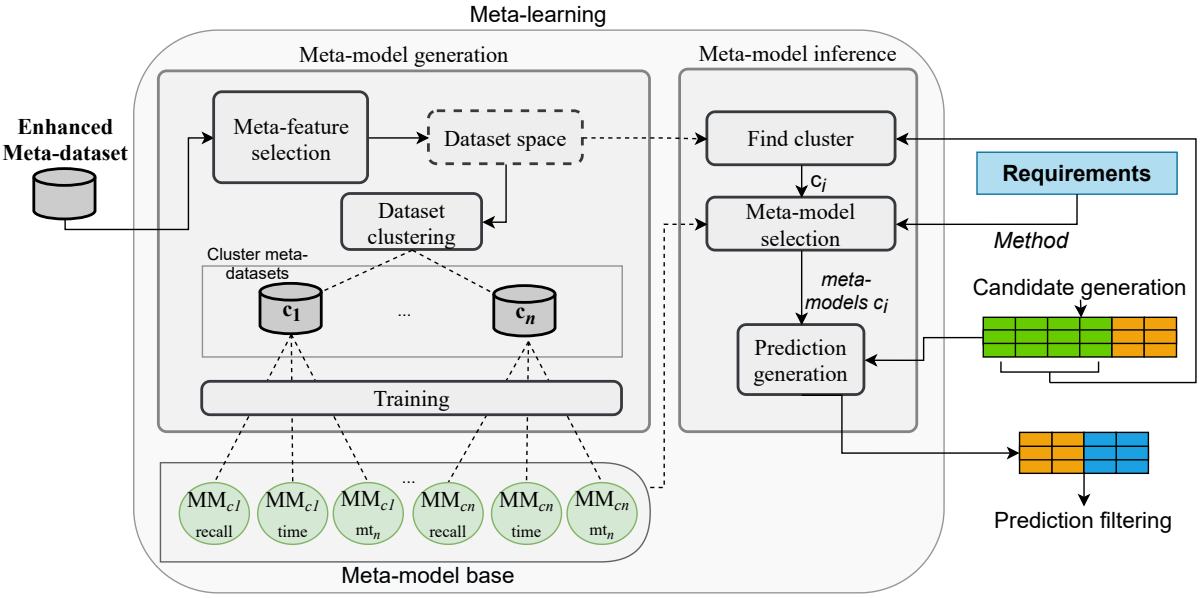


Figure 13 – The components and task flow of the proposed dataset-similarity-based recommender instantiation.

6.1.1 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 subsection 6.3.3). We select the features whose importance is above a threshold. We identified that 0.9 is a suitable value for the threshold.

6.1.2 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) [106].

Although there are alternatives for this idea of training meta-models for similar datasets only, we opted for DBSCAN because it is a generic and robust strategy. For instance, if we employ a k-NN algorithm to select the k most similar datasets for inducing a meta-model, we would have two drawbacks. The first one is we would need to induce the

meta-models in the recommendation time since the datasets in the training meta-dataset would only be known given the input dataset. Hence, each new dataset would require an online training phase to provide recommendations. The second drawback is twofold: we would have to choose a proper k value, which requires tuning time, and a fixed k could not be the optimal value for all datasets. Therefore, employing DBSCAN is a better option because good parameter values can be defined more robustly.

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). Another option could be the HDBSCAN algorithm [107], which does not require a distance parameter. However, we verified HDBSCAN generates too large clusters for our purpose. 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 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, for each cluster, we employ the Random Forest as the base-learner to induce a meta-model for each meta-target. The instantiation uses the default hyper-parameters from Scikit-learn 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 parametrization.

Employing cluster-based meta-models can also be helpful to manage the evolution of the meta-dataset in the framework. Adding a few new datasets to the meta-dataset may 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 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. The framework's evolution should consider complete refreshes

over time.

6.1.3 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 and predicts all the performances (meta-targets) that the candidate graph configurations should achieve, and forwards them to the prediction filtering task.

6.2 Improvements to the Meta-Knowledge

In this section, we explain how the meta-knowledge was enhanced in comparison to the previous chapter. This process involved the employment of new meta-features and the data augmentation procedure. For generating meta-instances, the process was the same as described in Chapter 5.

6.2.1 New Complexity Meta-Features

We included in this instantiation new meta-features alongside the ones used in the global instantiation presented in the previous chapter. All of them refer to the complexity of the nearest neighbor problem, as follows.

- Relative Variance (RV) of the dataset;
- 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;
- 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

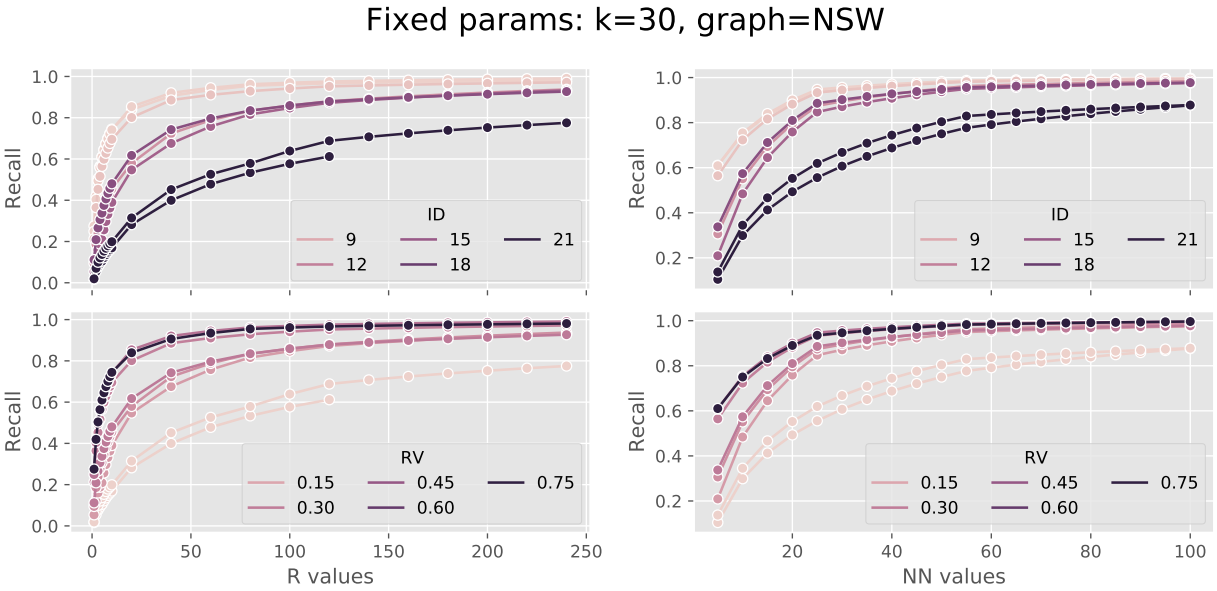


Figure 14 – Complexity meta-features that improve the measurement of hardness of datasets. This example regards all real datasets employed in this work for a fixed graph type and fixed k value. In the left we fixed $NN = 5$ and in the right $R = 5$. The first line refers to the intrinsic dimensionality, and the higher its value the harder the dataset is. The second line refers to the relative variance, and the lower its value the harder the dataset is.

(rv) was also estimated by sampling the same instances from the input dataset. The implementation we developed to generate these meta-features is available online[1].

We focused on meta-features related to the dataset complexity because this aspect is crucial for similarity retrieval parameter settings. For instance, Figure 14 shows the impact of the intrinsic dimensionality (ID) and the relative variance (RV) in query recall regarding the real datasets employed in this work. In this example, we selected the NSW graph, fixed the index parameter as $NN = 5$ (left), $R = 5$ (right), and the number of retrieved elements as $k = 30$, to see how the recall score of queries behaved by increasing the R and NN parameters. For ID, the higher its value, the harder it is to perform nearest neighbor queries on the dataset. On the other hand, for RV it is the opposite; the lower its value, the harder it is to perform nearest neighbor queries on the dataset. We can see that for easier datasets, it is possible to achieve high recall rates by increasing the R or NN values w.r.t. the fixed parameter. However, the same parameter values for these datasets cannot do the same for harder ones. This evaluation illustrates that the employed complexity meta-features are suitable to characterize our problem domain. Providing information about the hardness of performing nearest neighbor queries to our meta-model is essential to achieve good predictive performances.

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

6.2.2 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. Figure 15 shows an example of the original parameters values and the interpolated values. In the example, the NN is fixed, and the interpolation is performed over the R values. We employed the quickhull algorithm [108], 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. It can be seen that 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 significant variability for measures.

6.3 Analysis of the Instantiation's Supporting Techniques

In this section we intend to better explain how the techniques supporting the dataset-similarity-based instantiation behave and impact the performance of the approach.

We start by showing how the data augmentation was performed by interpolating meta-targets according each graph setting. We alternate the train and test sets to evaluate the impact of such evaluation in the final predictive performance. Further, we also discuss how the new meta-features and the new meta-instances improved the predictive performance over the time. Finally, we explain how the clustering procedure was performed, by first selecting relevant meta-features according to a specific meta-target.

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

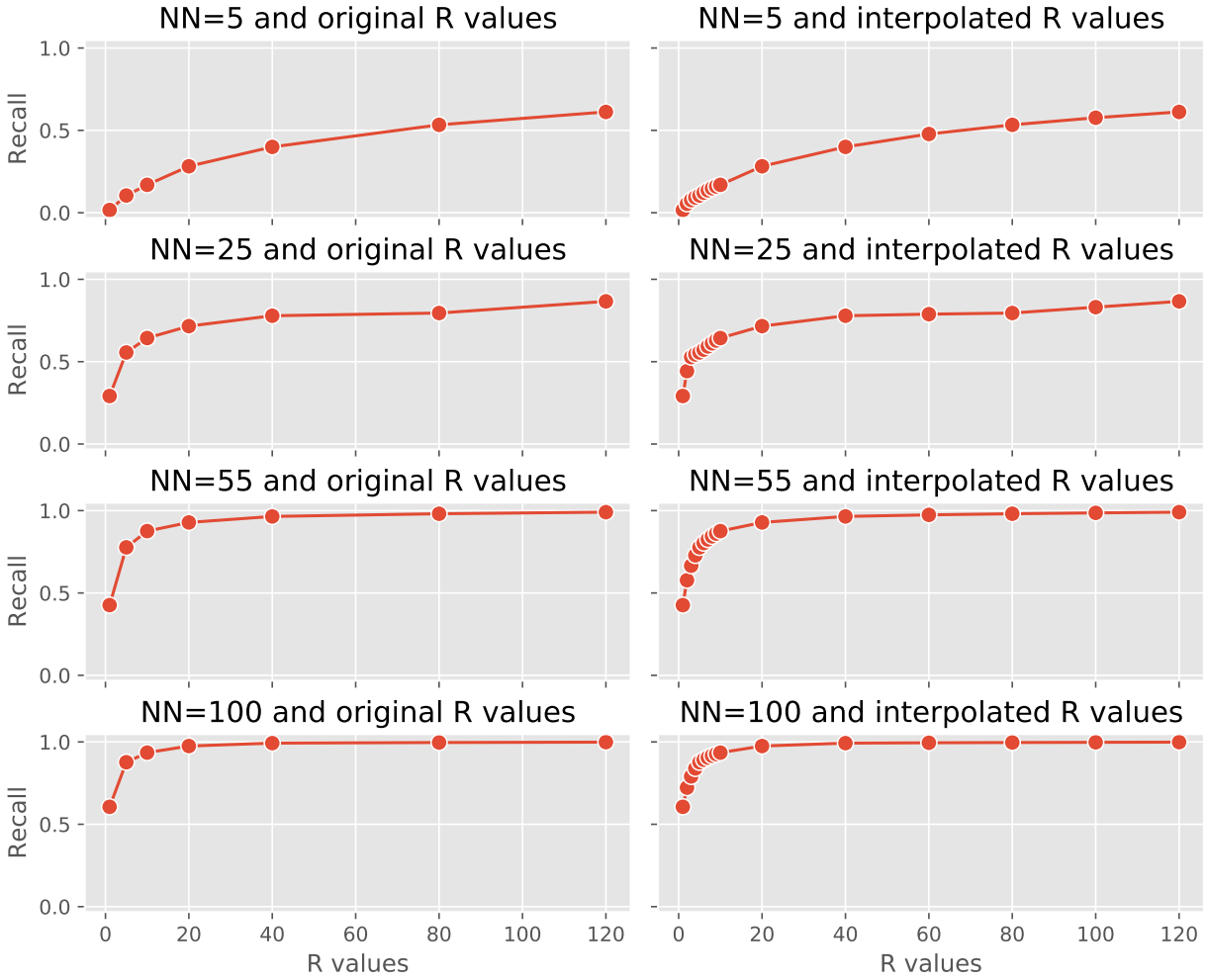


Figure 15 – Interpolation of a meta-target to generate weak-labeled meta-instances. The original values for the recall (left) and interpolated parameter values (right), for varying NN and R.

6.3.1 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 from the previous chapter to assess the overall predictive performances by varying the training and testing sets using data augmentation. We considered four different approaches, detailed following.

- Approach 1 (*a1*): limited parameter space for both training and testing.
- Approach 2 (*a2*): limited parameter space for training and interpolated parameter values for testing.
- Approach 3 (*a3*): interpolated parameter values for training and limited parameter space for testing.
- Approach 4 (*a4*): interpolated parameter values for both training and testing.

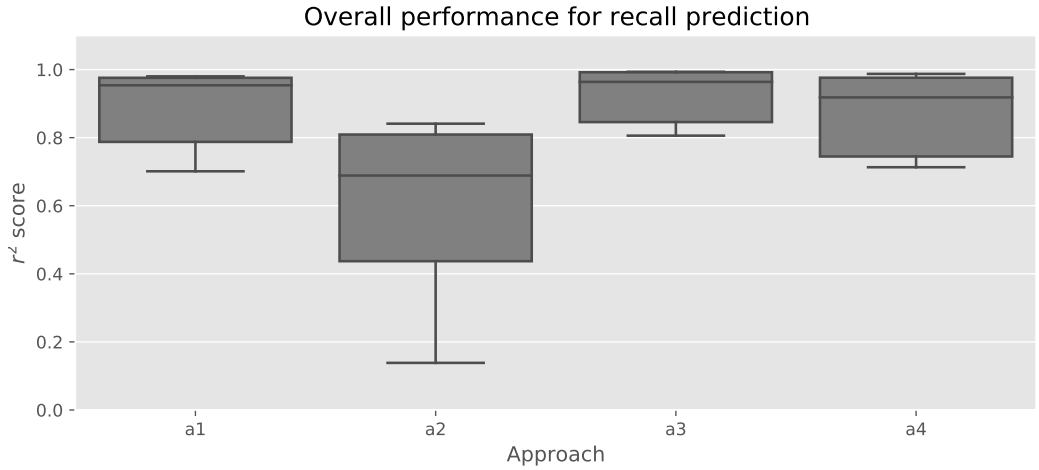


Figure 16 – Performance comparison using train/test procedures by alternating between the original and the augmented meta-datasets.

Limited parameter space refers to only using the original meta-instances, whose meta-targets were extracted from graph-based methods by running batches of experiments. Interpolated parameter values refer to meta-instances weak-labeled using interpolated meta-target values.

Figure 16 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 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 subsection 6.2.2. Comparing *a1* and *a3*, it is clear that the data augmentation improved the meta-models' performance as the testing set of both approaches is the same. Finally, analysing *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 leads to 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.

6.3.2 Impact of the New Meta-features and Datasets

Part of the dataset-similarity-based instantiation improvement comes from enhancing the meta-datasets with the proposed new meta-features and additional real image datasets. The previous chapter considered only the intrinsic dimension as a complexity

meta-feature. On the other hand, this chapter employs all meta-features described in Table 2, including the new complexity meta-features. Further, we included two new datasets to generate more meta-instances, *FashionMNIST* and *MNIST121d*. Both are variations from the *MNIST* dataset. *FashionMNIST*[3] has the same cardinality and dimensionality of *MNIST*, but it contains images that are considered more complex while *MNIST121d* is the original *MNIST* downscaled to 121 dimensions. Table 7 presents a new version of Table 4, showing the properties of all datasets, highlighting the new ones.

Table 7 – The real and synthetic datasets employed in this chapter and their characteristics.

	Title	**Size**	**Dimensions**
Real	*Moments*	68,040	9
	Texture	68,040	16
	Histogram	68,040	32
	MNIST	70,000	784
	SIFT	1,000,000	128
	MNIST121d	**70,000**	**121**
	FashionMNIST	**70,000**	**784**
	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 analyzed how the meta-dataset enhanced with the new meta-features and the new datasets impact the meta-models' predictive performance induced using different versions of the meta-dataset. 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.

- *MF_V1 + MI_V1*: regards the meta-features and datasets from the previous chapter;
- *MF_V1 + MI_V2*: uses the meta-features from the previous chapter and all datasets, including the new ones (*FashionMNIST* and *MNIST121d*);
- *MF_V2 + MI_V1*: uses the complete meta-feature set, including the new meta-features presented in this chapter and only the datasets from the previous chapter;
- *MF_V2 + MI_V2*: regards the complete meta-feature set and all datasets.

Figure 17 shows the achieved GMM's r^2 performances. We can notice a meaningful improvement at each enhancement. Either for only adding new meta-features or for only adding meta-instances from new datasets, the improvement in the r^2 score is evident.

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

Nonetheless, we achieved the highest average performance with the lowest variation by employing both new meta-features and meta-instances from new datasets. These results confirm that the improvements we proposed to the meta-knowledge contribute to getting higher average predictive performances, leading to improved recommendations. From now on, we consider the enhanced meta-dataset using the complete meta-feature set and all datasets.

6.3.3 Impact of Feature Selection Techniques on the Dataset-similarity-based Meta-Models

This section explains the impact of traditional feature selection methods employed to define the similarity space for clustering the datasets. We employ 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. For these two approaches, we consider the absolute value calculated for each meta-feature 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 propose in subsection 6.1.2, 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

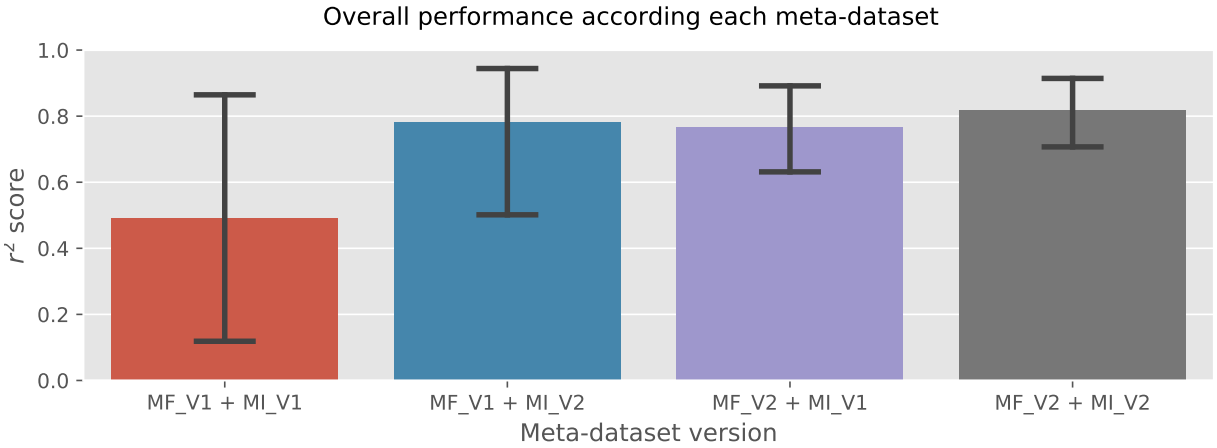


Figure 17 – Enhancement promoted by adding new meta-features and/or meta-instances from new datasets.

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.

Table 8 – Cluster information on dataset spaces defined by the feature selection methods.

FS method	Meta-Target	n clusters	outliers(%)	mean.mean	std.mean	max.mean	min.mean
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

We summarize the obtained information about this procedure in Table 8. 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 statistical metrics mean, standard deviation, maximum, and minimum. 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 338 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 cluster-based meta-models' performance 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 Figure 18. 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 using PCA, the results were wretched. 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 Figure 18 and the box-plot of the approach *a4* in Figure 16. These two box-plots refer to the average performance of the model(s) regarding recall for the same datasets. The box-plots show the superiority of the cluster-based meta-models compared to the global meta-model for this meta-target.

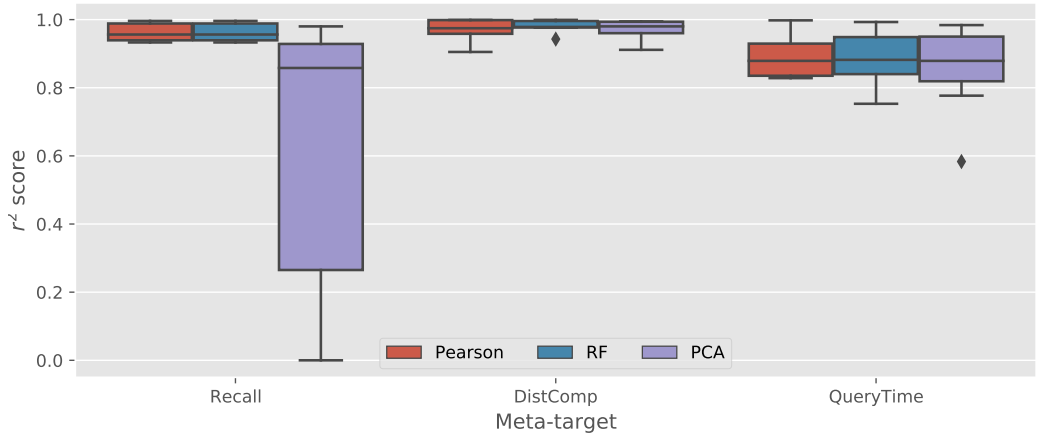


Figure 18 – (Meta-)Feature selection method considering the eps selected by the previous section.

Figure 19 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 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.

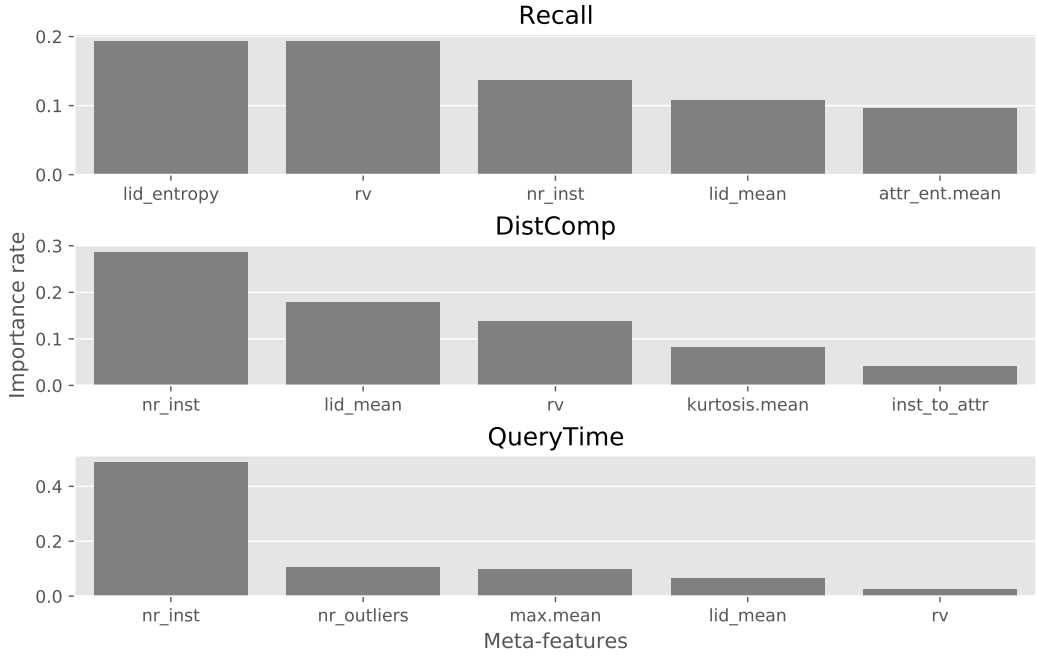


Figure 19 – The meta-features defining each meta-target's dataset space according to the RF method and their importances.

6.4 Analysis of the Instantiation's Prediction Accurracy and Execution Time

In this section we present the results regarding methods following the dataset-similarity-based recommender instantiation. We developed a generic and tuned methods for our recommendation framework (see subsection 4.4.1) following this instantiation strategy as we did for the global instantiation. Essentialy, we changed the strategy for training the meta-models, generating a set of meta-models per cluster. However, we maintained the essence of the generic and tuned methods, keeping the fine-tuning approach the same. Thus, we named the new methods by adding a "plus" at the end of the name of corresponding one, being *GMM+, TMM-GS+, TMM-S+*. Here, we provide experimental results about the predicion accuracy of the generated meta-models. We included the number of distance computations as a new meta-target. We also show a comparison between 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.

6.4.1 Prediction Accuracy of the Dataset-Similarity-based Meta-Models

In this section, we analyze the new meta-models performances. Figure 20 shows the r^2 scores achieved by the *GMM+*, *TMM-GS+* and *TMM-S+* methods, for each meta-target and each real dataset. The best overall performance was achieved when predicting the number of distance computations. For recall and distance computations, although the tuned methods increase the performance, *GMM+* was competitive. On the other hamd, the query time prediction was considerably improved by the fine-tuning methods. We can notice that for *Texture*, *FashionMNIST*, and *Histogram*, *TMM-S+* had an increase greater than 20%.

The most significant achievement of the new methods is a remarkable improvement in the recall prediction. For the recall prediction, the dataset-similarity-based methods had the most significant improvement when 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. For the other meta-targets, although sometimes the predictions are innacurate, the predicted values follow an order similar the order of the actual values. Providing an accurate order is fundamental for the ranking step in the framework's prediction filtering task. Therefore, the predictions provided by the cluster-based models allow generating high-quality parameter recommendations.

6.4.2 Prediction Accuracy and Recommendation Time of the Global and Dataset-Similarity-based Methods

Here we compare the predictive performance and execution time improvements for providing recommendations of the global and the dataset-similarity-based methods.

Figure 21 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, in order to simulate a scenario where each of them would be an unkown dataset. We can notice significant performance increases of all the dataset-similarity-based methods, with exception of the *GMM+* for query time prediction. The most significant improvement was the *GMM+*'s recall prediction compared to the *GMM*'s. Therefore, the ideal scenario would be *GMM+* always providing the best recommendations. The advantage here is due to the fact that this method has no fine-tuning, thus the time for recommendation is only the meta-model inference, which is fast. However, this is utopic as datasets will always have different properties in real world. We can feed our meta-model with a huge amount of different datasets characteristics, but there is no guarantee that it would be generic enough. In this sense, it is noticeable in Figure 21 that the fine-tuning approaches remain important for improving the quality of predictions of the dataset-similarity-based methods.

We measured the time needed to generate a recommendation to show the effort to invest to get better recommendations. Figure 22 shows the elapsed time for getting

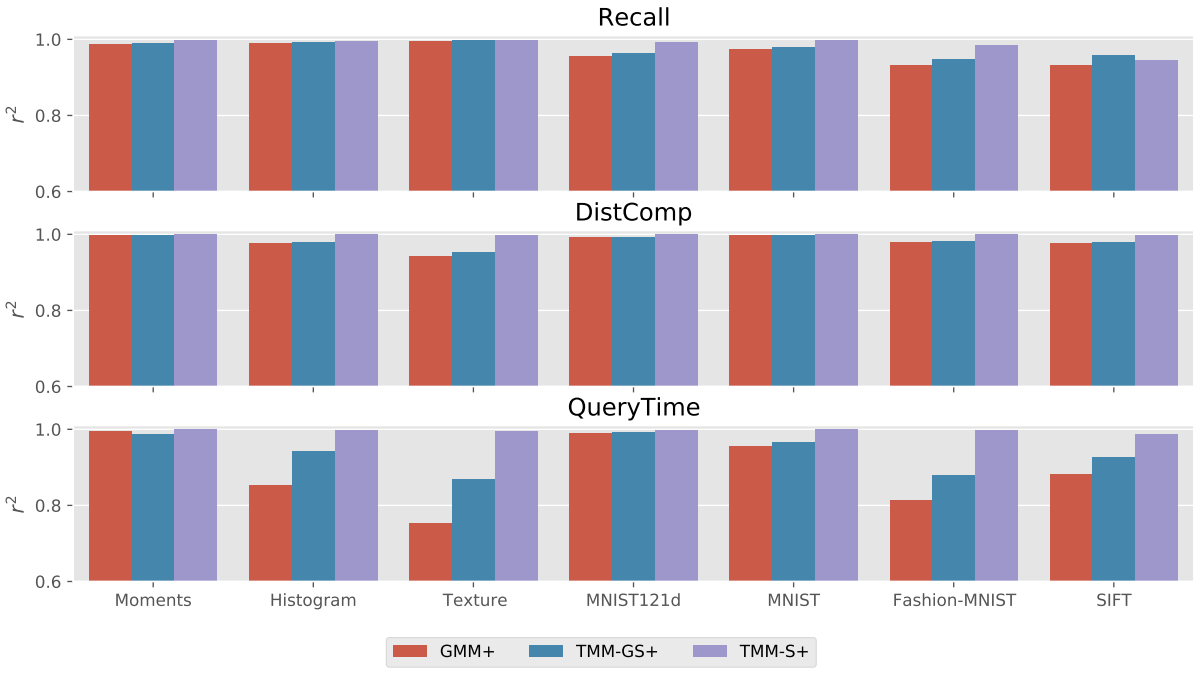


Figure 20 – r^2 scores achived by each dataset-similarity-based meta-model, regarding each meta-target and each dataset.

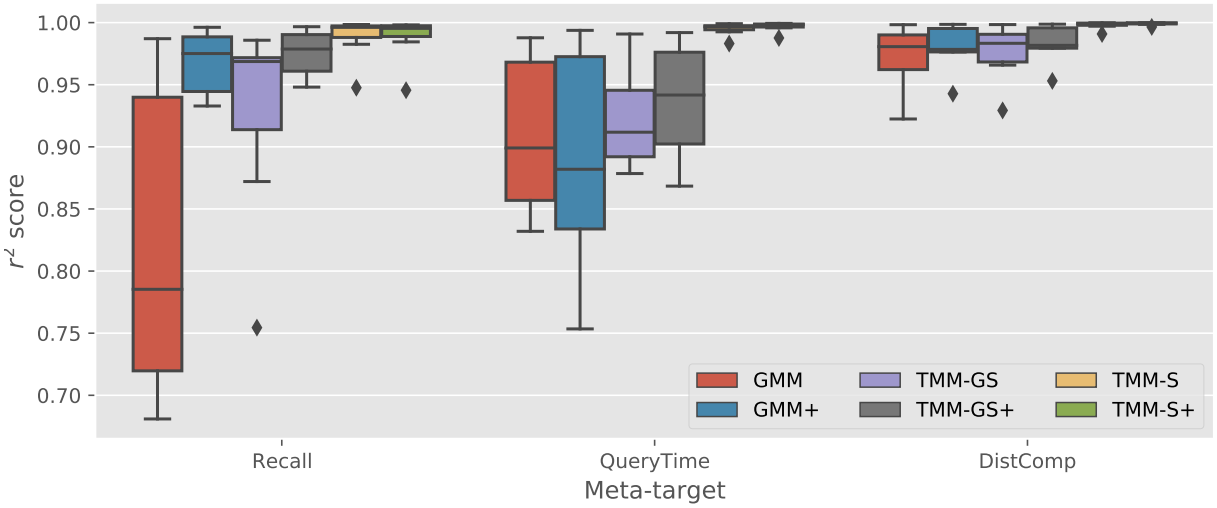


Figure 21 – Predictive performance distribution regarding each method and meta-target for all datasets.

recommendations according 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 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 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 and the trainig 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-*

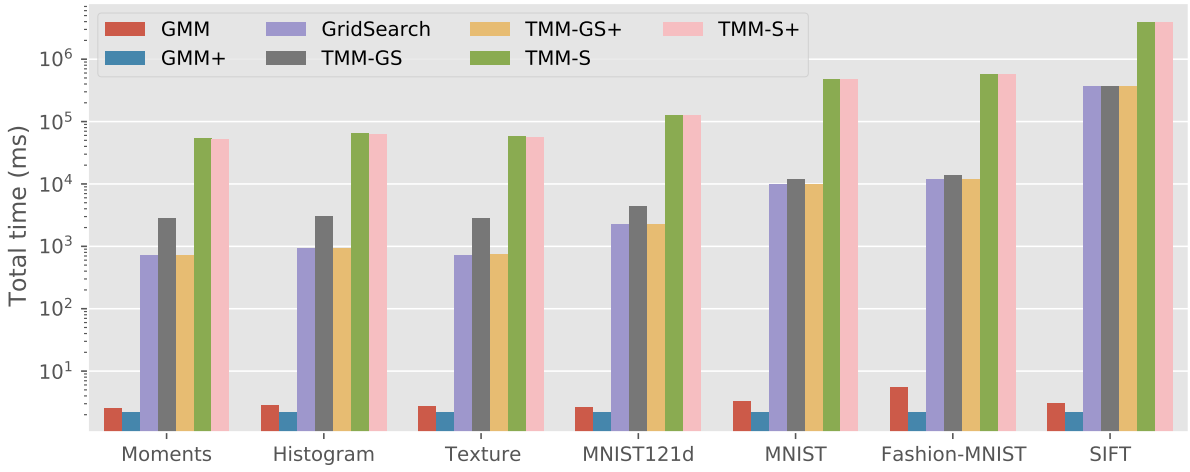


Figure 22 – The elapsed time for providing the final predictions for all meta-targets according to each method and dataset.

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 certain cluster in *TMM-GS+*, this time is significantly lower. The execution time of tuning using subsets has the same behaviour as tuning using grid search. However, 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.

Clearly, the tuning procedures are highly time consuming. However, it may be worth the tuning effort depending on the complexity of the input dataset. We present results regarding the final recommendations of each method for each real dataset in the next section.

6.5 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 quick methods and the latter for tuned methods. Both parts are compared with the optimal cases, which are the graph configurations that achieved the best performance, satisfying the minimum recall constraint.

Quick methods do not require running online experiments on the input dataset to generate meta-targets. *GMM* and *GMM+* are in this category. We established two methods to serve as a quick method baseline, named as *Tight* and *Loose*, detailed in subsection 6.5.1. These two baselines simulate a generic recommendation provided by an expert regardless of the input dataset. Tuned methods require investing time to extract meta-targets from the input dataset, namely *TMM-GS*, *TMM-GS+*, *TMM-S*, *TMM-S+*, and Grid Search as a baseline.

We considered three optimization metrics: the query time, the memory usage, and the number of distance computations. We present the results regarding each of these metrics by considering different required recall values: 90%, 95%, and 99%. 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 recommendations because the predicted recall is above the requirement, but its true value is not.

6.5.1 Quick Methods

This section presents the quick methods' effectiveness, comparing the *GMM* and *GMM+* methods with the optimal results and two baselines.

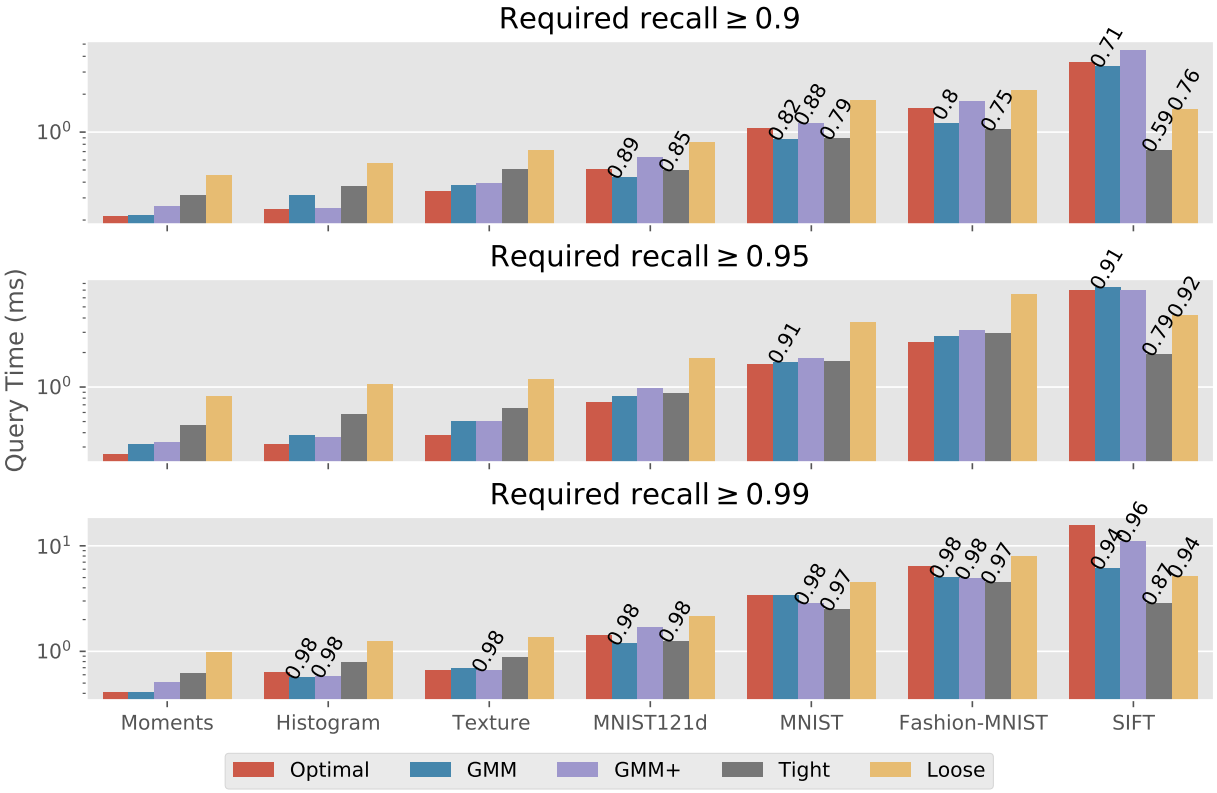


Figure 23 – Recommendations optimizing the query time regarding the quick methods.

The baselines *Tight* and *Loose* are defined by averaging the parameter values regarding the top-10% configurations (over our entire result set, including 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.

- *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.

Figure 23 shows recommendations optimizing query time. Considering required recall values less than 0.99, both proposed methods *GMM* and *GMM+* outperformed the baselines, in general. In these cases, *GMM+* performed accurately, frequently rea-

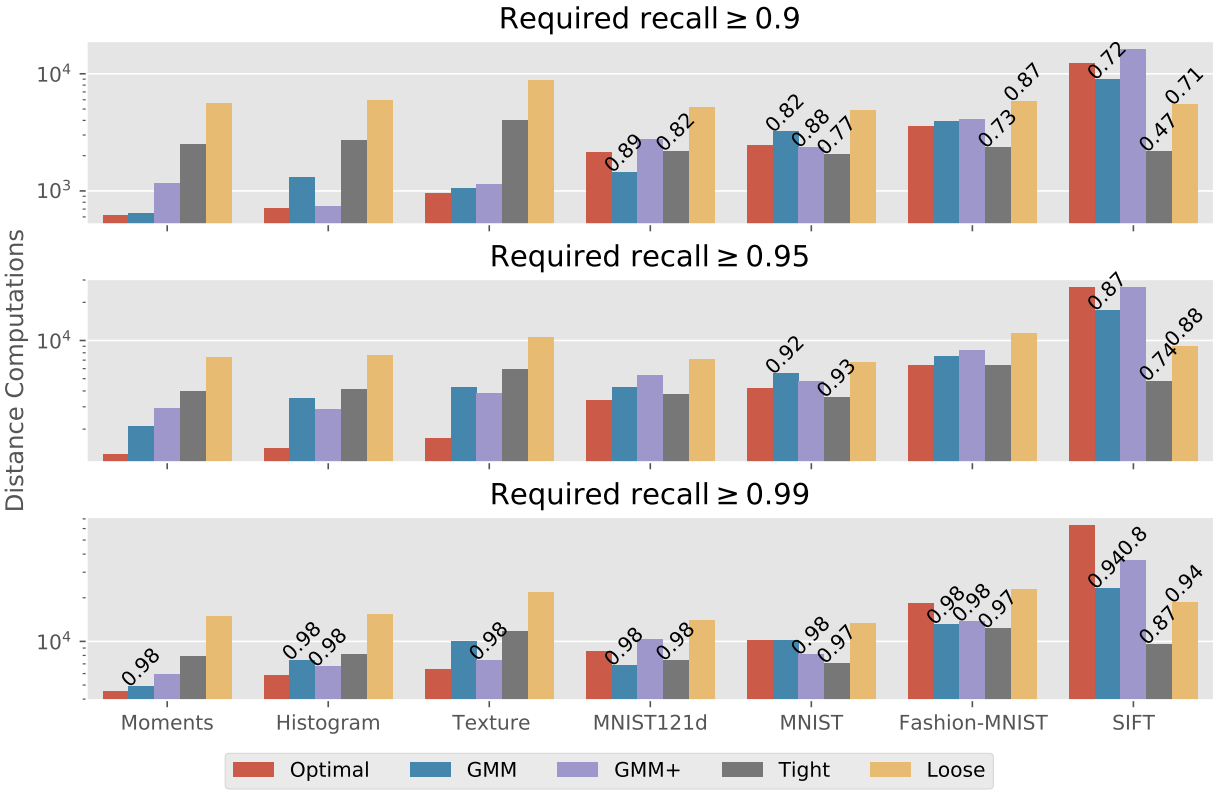


Figure 24 – Recommendations optimizing the number of distance computations regarding the quick methods.

ching the optimal or getting very close to it. For recall ≥ 0.99, *GMM* presented the best recommendations. All methods generated wrong recommendations, being the most significant differences between the predicted and the required recall values noticed for *SIFT* with recall ≥ 0.99. The *Loose* recommendation rarely provides wrong recommendations; however, it is always far from the optimal recommendation. *GMM+* was the best method as it provided a little more wrong recommendations than *Loose*, but always with actual recall very close to the required. On the other hand, *GMM* and *Tight* provided wrong recommendations more often and presenting a higher recall error.

Considering recommendations optimizing distance computations, in Figure 24, *GMM+* performed very well for required recall values less than 0.99, providing only one wrong recommendation. In general, *GMM* and *GMM+* recommended well, alternating the best recommendation. For recall ≥ 0.99, the error rate was diminutive regarding both methods, except for *SIFT*. Nonetheless, again, the error rate of wrong recommendations provided by *GMM* was higher than the rate of *GMM+*. The only cases where the optimal was reached were for *Histogram* with required recall ≥ 0.9 and for *SIFT* with required recall ≥ 0.95 by *GMM+*; and for *MNIST* with required recall *ge* 0.99 by *GMM*. *GMM+* was again the most effective because it provided fewer wrong recommendations than *GMM* and *Tight* and better recommendations than *Loose*.

Lastly, Figure 25 presents recommendations optimizing the memory usage. In this

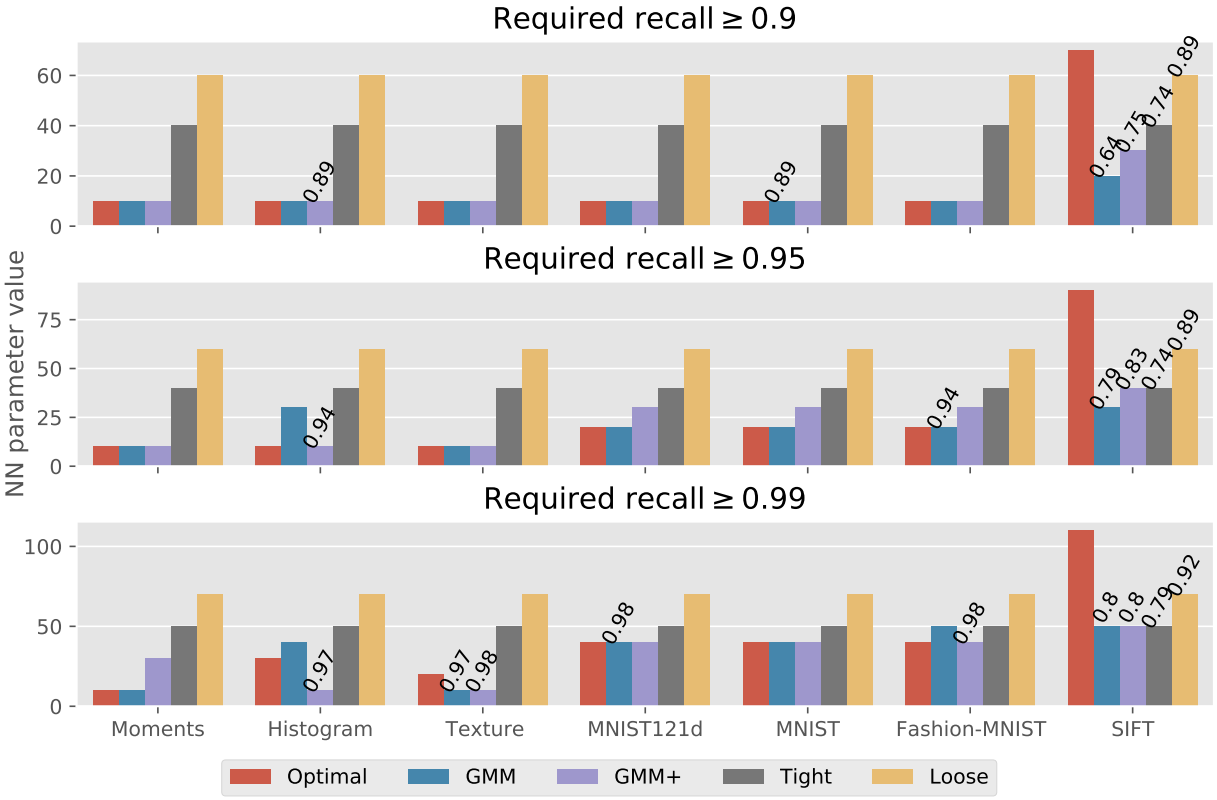


Figure 25 – Recommendations optimizing memory usage regarding the quick methods. This metric is estimated by the number of edges (*NN* parameter) of the graph. The higher this value, the higher the memory consumption.

case, we had poor recommendations for the *SIFT* dataset, where all methods failed. For the remaining datasets and all recall values, our methods significantly outperformed the baselines, except for *FashionMNIST* with recall ≥ 0.99 where *Tight* tied with *GMM*. They provided a few wrong recommendations but with minor differences between the required recall and the actual recall. We can also notice that *GMM* and *GMM+* were able to frequently reach the optimal cases, being the winning options for optimizing memory usage.

6.5.2 Tuned Methods

This section presents the results achieved by the methods using tuning. We employ the classic Grid Search as a baseline and compare it with all tuned methods proposed in this work: *TMM-GS*, *TMM-GS+*, *TMM-S*, and *TMM-S+*. The main objective of our tuned methods is to invest effort before recommending to increase the predictive performance and, consequently, improve final recommendations.

Figure 26 shows the recommendations optimizing query time. Firstly, notice that for recommendations for a recall less than 0.99, we had fewer cases of wrong recommendations compared to the quick methods. For recall ≥ 0.99, we still had a high number of unsuitable recommendations; however, the error was relatively low, limited to 0.01 in

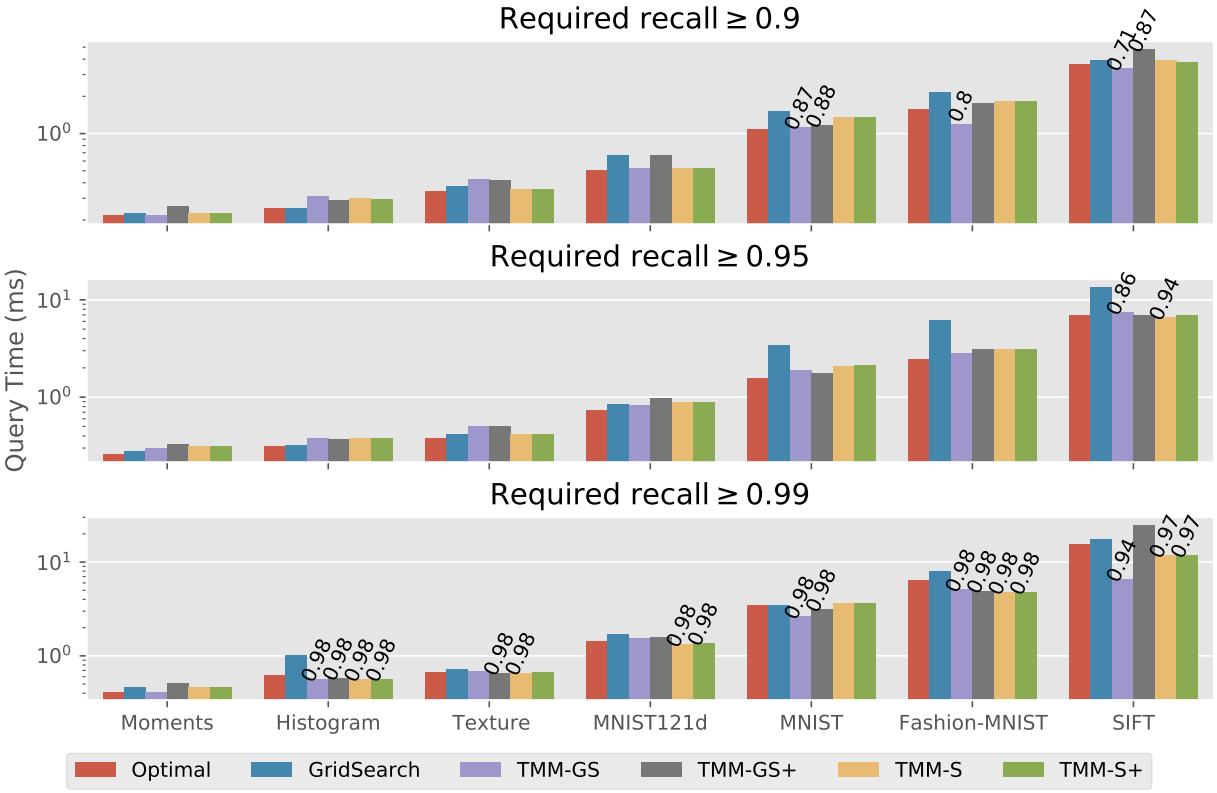


Figure 26 – Recommendations optimizing the query time regarding the tuned methods.

most cases. Regarding the *TMM-GS* and *TMM-GS+* methods, the former provided wrong recommendations more often, although both methods performed similarly for correct recommendations. On the other hand, the *TMM-S* and *TMM-S+* had almost the same performance, with a slight difference for *Histogram* with recall ≥ 0.9, *MNIST* with recall ≥ 0.95, and *SIFT* with recall values less than 0.99, where *TMM-S+* was superior. The Grid Search strategy never provided wrong recommendations. Nevertheless, one or more of the proposed tuned methods often outperformed Grid Search. Overall, *TMM-S+* the method closest to the optimal case.

The recommendations optimizing the number of distance computations showed in Figure 27 presented a behavior similar to the behavior optimizing the query time. For required recall values in [0.90, 0.95], our methods provided only a few wrong recommendations. On the other hand, the number of while for recall ≥ 0.99 our methods often provided wrong recommendations, with error rates significantly low. The Grid Search was the best for the easier datasets and lower recall rates, while our methods win for the other cases. *TMM-GS* and *TMM-GS+* performed alike, however the error rate for *TMM-GS+* was lower. Again, *TMM-S* and *TMM-S+* performed close to each other, being *TMM-S+* slightly better.

Lastly, for recommendations optimizing memory usage, presented in Figure 28, we can outstand our methods frequently reaching the optimal, except for *SIFT*. For the other datasets, when we had wrong recommendations, the error rates were considerably

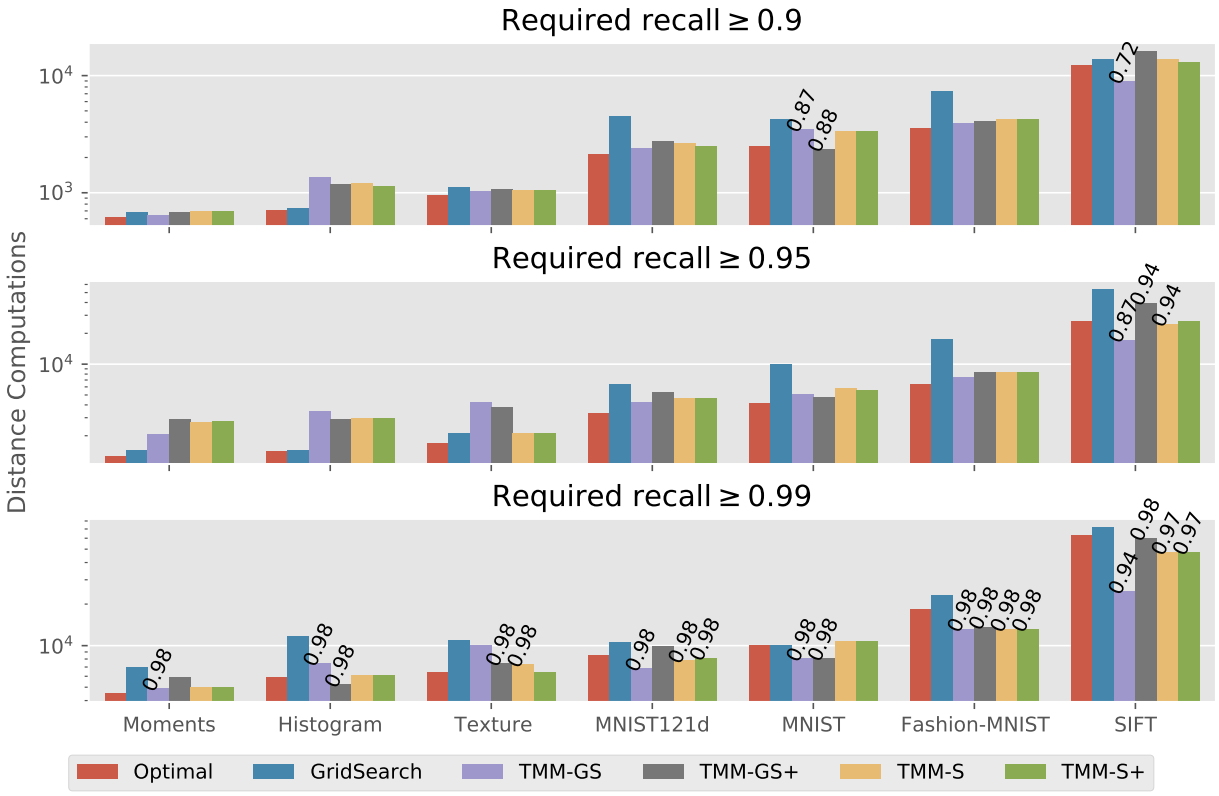


Figure 27 – Recommendations optimizing the number of distance computations regarding the tuned methods.

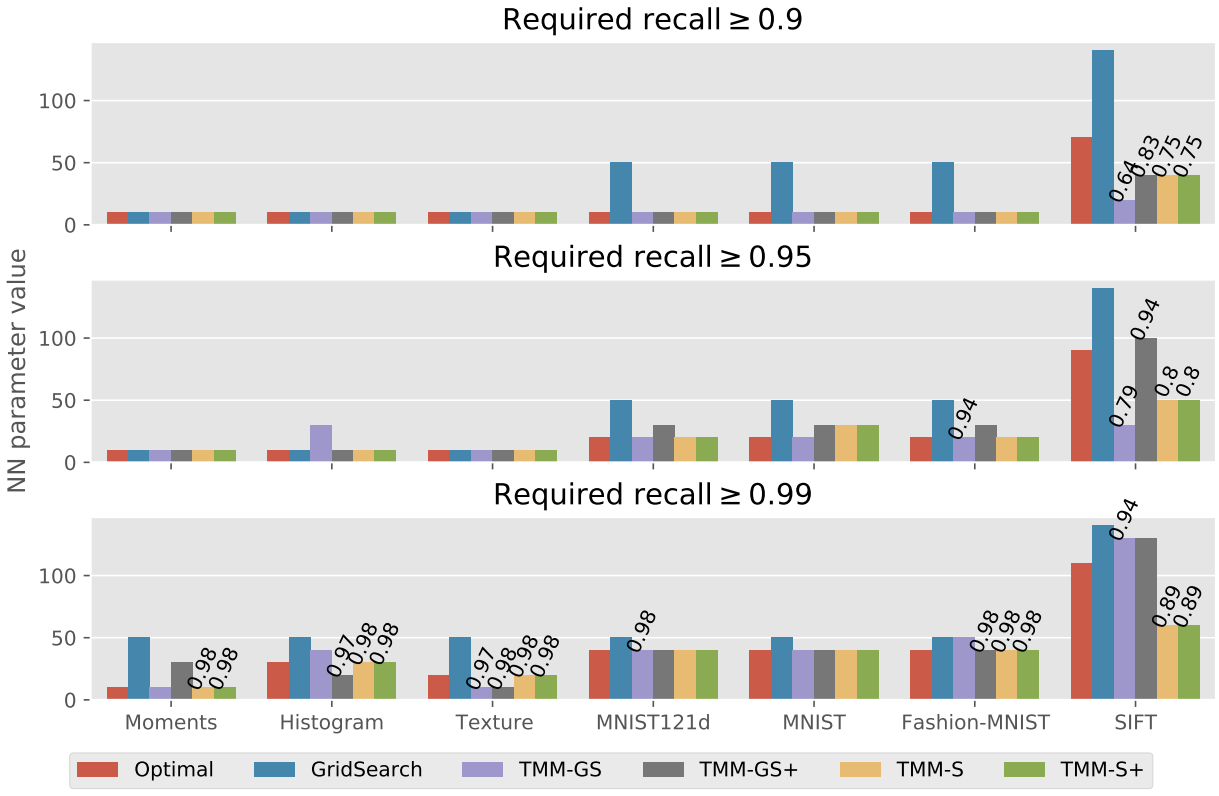


Figure 28 – Recommendations optimizing memory usage regarding the tuned methods.

low. Surprisingly, the only correct recommendation for this dataset provided from one of our methods was with recall ≥ 0.99, by *TMM-GS+*. Nevertheless, we can highlight that our methods outperformed the Grid Search in all cases for *MNIST* and its variations, which have the highest dimensionalities in this work.

This chapter presented a detailed explanation and experimentation about the proposed database-similarity-based recommender instantiation and the global and tuned methods following this instantiation. The results showed we achieved improved predictive performance, allowing our system to provide more robust recommendations. A remarkable achievement here was the performance of the *GMM+* as it is the fastest method proposed in this work. It outperformed fixed general recommendations in most cases. Even in cases where *GMM+* produced wrong recommendations, the difference between the required recall and the true value achieved by the recommendation was quite low, in general. We also highlight that our tuned methods provided the overall best recommendations, being the closest to the optimal recommendations. These methods achieved the optimal many times and frequently outperformed the *Grid Search* regarding all optimization metrics. However, we have room for improvements to our methods regarding recommendations satisfying a high recall (e.g., 0.99), where the acceptable error is tiny. The results corroborate that this work proposed a framework and instantiation strategies that advance the state-of-the-art regarding the problem of selecting proximity graphs and their parameters.

7 CONCLUSION

In this work, we proposed an intelligent system capable of recommending a graph-based method and its configurations to perform similarity searches, given a dataset, user-defined constraints, and an optimization criteria. The main idea consists of using meta-learning techniques to characterize datasets and estimate the relative performance of different graph configurations on them. We focused on recommendations for image databases and presented two general versions of meta-models using Random Forests. We also proposed three fine-tuning approaches for these meta-models instantiations, *GMM*, *TMM-GS* and *TMM-S*, and compared them with baselines simulating expert users and with a standard grid search strategy.

7.1 Discussion

We initially showed in Chapter 5 that it is possible to develop a robust recommendation system by employing simple meta-features to discriminate ordinary datasets, but for more difficult cases it is necessary to expand the meta-knowledge space by investing a certain amount of time to generate more meta-features from similar datasets.

Results from this first version showed that, for many situations, the generic approach (*GMM*) performed similarly or outperformed the grid search, without requiring the user to execute performance measures on the goal dataset. However, it failed to provide valid recommendations for the most complex datasets tested. Thus, the first problem that we outlined was the prediction performance of our generic meta-model in these cases. We believe that when our *GMM* performed poorly, the reason was not the complexity of the datasets, but the lack of datasets with similar properties in the meta-knowledge. This limitation was notoriously addressed by tuning the meta-model with subsets of the input dataset, which means that if we increase the meta-knowledge with datasets that present similar characteristics to the input, we tend to generalize our recommendation system and, consequently, accomplish better recommendations. On the other hand, such an enhanced approach, *TMM-S*, reached optimal results, but it is an expensive approach. Therefore, as an initial improvement, we aimed at developing a new method to perform as well as the *TMM-S* but without being high-time consuming

We presumed that for achieving this goal, we could essentially focus on three tasks: exploring new relevant dataset descriptors regarding the nearest neighbor problem, in order to better discriminate them; enhancing our meta-dataset by running batteries of experiments to generate meta-instances from new datasets and performing an augmentation procedure via interpolation; and lastly, proposing a new strategy for instantiating

the meta-model to achieve a better generalization. In Chapter 6 we presented a detailed description of each of these tasks, and how the meta-model performance was affected. Subsequently, we also presented results showing the recommendation effectiveness of our proposed methods.

We discussed step by step about these tasks showing how effective each phase was. The experiment combining the meta-dataset with old/new meta-features and old/new datasets supported the claim that enhancing the meta-knowledge implies higher predictive performances. Clearly, by adding the *MNIST121d* and *FashionMNIST* datasets, we achieved a better generalization, allowing the newly proposed methods to perform better for the original *MNIST* compared to older ones. We intuitively opted for not including similar datasets to *SIFT*, in order to understand how our proposed methods would behave in the presence of an "outlier" and how much the synthetic datasets were able to enhance our meta-knowledge. Although these procedures of enhancement were very efficient, the new strategy of evaluating the similarity among datasets also had an important role in the improvement of this work. We separated this discussion into quick and tuned methods.

For quick methods, we included those that did not require any extra time for the recommendation. Thus, we proposed two baselines simulating an expert user to compare with our methods, attempting to provide a *Tight* recommendation, which regards a more precise and adjusted graph configuration, and a *Loose* recommendation, which presumes that the graph configuration will at least satisfy the given requirement. The overall idea of these baselines is to propose a fixed graph configuration for each required recall value and for each optimization metric without knowing the input dataset, based on all results regarding the graph performances obtained through this work. In a few cases, the *Tight* method outperformed some of our methods, but in general, the *GMM* and *GMM+* performed better than both baselines, frequently reaching close values to the optimal with respect to all optimization metrics.

On the other hand, the presented methods which need an investment of a certain amount of time to acquire recommendations outperformed the *Grid Search* for more complex datasets. For the easier ones, they performed similarly. The *SIFT* was the most problematic dataset employed in this work for all proposed methods in general, but the *TMM-S* and *TMM-S+* were able to achieve reasonable recommendations, reaching the optimal in some cases. Regarding the correct recommendations, all tuned methods outperformed the *Grid Search* for *MNIST* and its variations for all required recall values and all optimization metrics. Similarly to the *GMM+*, the wrong recommendations presented a small difference between the recommended and the required recall, with respect to the highest required recall value evaluated in this work. This is remarkable as even if the meta-model has predicted wrongly, the error was not very keen.

7.2 Future Work

As future work, we intend to address to the following ideas:

- Find a strategy to decide whether an input dataset would require a tuning procedure or the generic meta-model would be robust enough to provide suitable recommendations;
- Employ our proposed framework for different index structures, such as hash-based or tree-based methods;
- Explore more metrics of "hardness" of datasets regarding the nearest neighbor problem, as it is being continuously studied in the literature;
- Expand the meta-dataset intelligently. That is, we must add new datasets towards a better generalization. Adding datasets with well-known properties (e.g. similar to *Texture* and *Moments*) would be worthless.

REFERENCES

[1] FALCONER, K. *Fractal geometry - mathematical foundations and applications*. [S.l.]: Wiley, 1990. ISBN 978-0-471-92287-2.

[2] ZEZULA, P. et al. *Similarity Search - The Metric Space Approach*. [S.l.]: Kluwer, 2006. v. 32. (Advances in Database Systems, v. 32). ISBN 978-0-387-29146-8.

[3] NAVARRO, G. Searching in metric spaces by spatial approximation. *VLDB J.*, v. 11, n. 1, p. 28–46, 2002.

[4] JR., C. T. et al. Slim-trees: High performance metric trees minimizing overlap between nodes. In: *EDBT*. [S.l.]: Springer, 2000. (Lecture Notes in Computer Science, v. 1777), p. 51–65.

[5] VIEIRA, M. R. et al. DBM-Tree: A dynamic metric access method sensitive to local density data. In: CITESEER. *In SBBD*. [S.l.], 2004.

[6] NAVARRO, G.; REYES, N. Dynamic spatial approximation trees for massive data. In: SKOPAL, T.; ZEZULA, P. (Ed.). *SISAP*. [S.l.]: IEEE Computer Society, 2009. p. 81–88.

[7] CHEN, L. et al. Efficient metric indexing for similarity search and similarity joins. *IEEE Trans. Knowl. Data Eng.*, v. 29, n. 3, p. 556–571, 2017.

[8] INDYK, P.; MOTWANI, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In: VITTER, J. S. (Ed.). *STOC*. [S.l.]: ACM, 1998. p. 604–613.

[9] ZHANG, Y. et al. Fast *kNN* graph construction with locality sensitive hashing. In: BLOCKEEL, H. et al. (Ed.). *ECML PKDD*. [S.l.]: Springer, 2013. (Lecture Notes in Computer Science, v. 8189), p. 660–674.

[10] WANG, J. et al. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[11] LIU, W. et al. I-LSH: I/O efficient c-approximate nearest neighbor search in high-dimensional space. In: *IEEE ICDE*. [S.l.]: IEEE, 2019. p. 1670–1673.

[12] JAFARI, O. et al. A survey on locality sensitive hashing algorithms and their applications. *CoRR*, abs/2102.08942, 2021.

[13] ESULI, A. Use of permutation prefixes for efficient and scalable approximate similarity search. *Inf. Process. Manag.*, v. 48, n. 5, p. 889–902, 2012.

[14] AMATO, G.; GENNARO, C.; SAVINO, P. MI-File: using inverted files for scalable approximate similarity search. *Multimedia Tools Appl.*, v. 71, n. 3, p. 1333–1362, 2014.

[15] NAIDAN, B.; BOYTSOV, L.; NYBERG, E. Permutation search methods are efficient, yet faster search is possible. *PVLDB*, v. 8, n. 12, p. 1618–1629, 2015.

[16] NOVAK, D.; ZEZULA, P. PPP-Codes for large-scale similarity searching. *Trans. Large Scale Data Knowl. Centered Syst.*, v. 24, p. 61–87, 2016.

[17] FIGUEROA, K.; PAREDES, R.; REYES, N. New permutation dissimilarity measures for proximity searching. In: MARCHAND-MAILLET, S.; SILVA, Y. N.; CHÁVEZ, E. (Ed.). *SISAP*. [S.l.]: Springer, 2018. (Lecture Notes in Computer Science, v. 11223), p. 122–133.

[18] JAROMCZYK, J. W.; TOUSSAINT, G. T. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, v. 80, n. 9, p. 1502–1517, 1992.

[19] PAREDES, R.; CHÁVEZ, E. Using the k-Nearest Neighbor Graph for proximity searching in metric spaces. In: CONSENS, M. P.; NAVARRO, G. (Ed.). *SPIRE*. [S.l.]: Springer, 2005. (Lecture Notes in Computer Science, v. 3772), p. 127–138.

[20] MALKOV, Y. et al. Approximate nearest neighbor algorithm based on navigable small world graphs. *Inf. Syst.*, v. 45, p. 61–68, 2014.

[21] IWASAKI, M.; MIYAZAKI, D. Optimization of indexing based on k-Nearest Neighbor Graph for proximity search in high-dimensional data. *CoRR*, abs/1810.07355, 2018.

[22] FU, C. et al. Fast approximate nearest neighbor search with the Navigating Spreading-Out graph. *Proc. VLDB Endow.*, v. 12, n. 5, p. 461–474, 2019.

[23] SHIMOMURA, L. C. et al. A survey on graph-based methods for similarity searches in metric spaces. *Inf. Syst.*, v. 95, p. 101507, 2021.

[24] HAJEBI, K. et al. Fast approximate nearest-neighbor search with k-Nearest Neighbor Graph. In: WALSH, T. (Ed.). *IJCAI*. [S.l.]: IJCAI/AAAI, 2011. p. 1312–1317.

[25] AUMÜLLER, M.; BERNHARDSSON, E.; FAITHFULL, A. J. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.*, v. 87, 2020.

[26] BELUSSI, A.; FALOUTSOS, C. Estimating the selectivity of spatial queries using the "correlation" fractal dimension. In: DAYAL, U.; GRAY, P. M. D.; NISHIO, S. (Ed.). *VLDB*. [S.l.]: Morgan Kaufmann, 1995. p. 299–310.

[27] BEYER, K. S. et al. When is "nearest neighbor" meaningful? In: BEERI, C.; BUNEMAN, P. (Ed.). *ICDT*. [S.l.]: Springer, 1999. (Lecture Notes in Computer Science, v. 1540), p. 217–235.

[28] PAGEL, B.; KORN, F.; FALOUTSOS, C. Deflating the dimensionality curse using multiple fractal dimensions. In: LOMET, D. B.; WEIKUM, G. (Ed.). *ICDE*. [S.l.]: IEEE Computer Society, 2000. p. 589–598.

[29] KORN, F.; PAGEL, B. .; FALOUTSOS, C. On the "dimensionality curse" and the "self-similarity blessing". *IEEE Transactions on Knowledge and Data Engineering*, p. 96–111, 2001. ISSN 2326-3865.

[30] LEVINA, E.; BICKEL, P. J. Maximum likelihood estimation of intrinsic dimension. In: *NIPS*. [S.l.: s.n.], 2004. p. 777–784.

[31] HOULE, M. E. Local intrinsic dimensionality I: an extreme-value-theoretic foundation for similarity applications. In: BEECKS, C. et al. (Ed.). *SISAP*. [S.l.]: Springer, 2017. (Lecture Notes in Computer Science, v. 10609), p. 64–79.

[32] HOULE, M. E.; ORIA, V.; WALI, A. M. Improving *k-NN* graph accuracy using local intrinsic dimensionality. In: BEECKS, C. et al. (Ed.). *SISAP*. [S.l.]: Springer, 2017. (Lecture Notes in Computer Science, v. 10609), p. 110–124.

[33] AUMÜLLER, M.; CECCARELLO, M. Benchmarking nearest neighbor search: Influence of local intrinsic dimensionality and result diversity in real-world datasets. In: *EDML SDM*. [S.l.]: CEUR-WS.org, 2019. (CEUR Workshop Proceedings, v. 2436), p. 14–23.

[34] FRANÇOIS, D.; WERTZ, V.; VERLEYSEN, M. The concentration of fractional distances. *IEEE Trans. Knowl. Data Eng.*, v. 19, n. 7, p. 873–886, 2007.

[35] HE, J.; KUMAR, S.; CHANG, S. On the difficulty of nearest neighbor search. In: *ICML*. [S.l.]: icml.cc / Omnipress, 2012.

[36] WANG, H. et al. A note on graph-based nearest neighbor search. *CoRR*, abs/2012.11083, 2020.

[37] THRUN, S.; PRATT, L. Y. Learning to learn: Introduction and overview. Springer, p. 3–17, 1998.

[38] BRAZDIL, P. et al. *Metalearning - Applications to Data Mining*. [S.l.]: Springer, 2009. (Cognitive Technologies). ISBN 978-3-540-73262-4.

[39] VANSCHOREN, J. Meta-learning: A survey. *CoRR*, abs/1810.03548, 2018.

[40] PRUDÊNCIO, R. B. C.; LUDERMIR, T. B. Meta-learning approaches to selecting time series models. *Neurocomputing*, v. 61, p. 121–137, 2004.

[41] MANTOVANI, R. G. et al. To tune or not to tune: Recommending when to adjust SVM hyper-parameters via meta-learning. In: *IJCNN*. [S.l.]: IEEE, 2015. p. 1–8.

[42] FERRARI, D. G.; CASTRO, L. N. de. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Inf. Sci.*, v. 301, p. 181–194, 2015.

[43] AGUIAR, G. J. et al. A meta-learning approach for selecting image segmentation algorithm. *Pattern Recognit. Lett.*, v. 128, p. 480–487, 2019.

[44] OYAMADA, R. S. et al. Towards proximity graph auto-configuration: An approach based on meta-learning. In: DARMONT, J.; NOVIKOV, B.; WREMBEL, R. (Ed.). *ADBIS*. [S.l.]: Springer, 2020. (Lecture Notes in Computer Science, v. 12245), p. 93–107.

[45] MUJA, M.; LOWE, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In: *VISAPP*. [S.l.]: INSTICC Press, 2009. p. 331–340.

[46] HYVÖNEN, V. et al. Fast nearest neighbor search through sparse random projections and voting. In: JOSHI, J. et al. (Ed.). *IEEE BigData*. [S.l.]: IEEE Computer Society, 2016. p. 881–888.

[47] OCSA, A.; BEDREGAL, C.; CUADROS-VARGAS, E. A new approach for similarity queries using neighborhood graphs. In: SILVA, A. S. da (Ed.). *SBBD*. [S.l.]: SBC, 2007. p. 131–142.

[48] FORTUNE, S. Voronoi diagrams and Delaunay triangulations. In: *Computing in Euclidean geometry*. [S.l.]: World Scientific, 1995. p. 225–265.

[49] WANG, J.; LI, S. Query-driven iterated neighborhood graph search for large scale indexing. In: BABAGUCHI, N. et al. (Ed.). *ACM MM*. [S.l.]: ACM, 2012. p. 179–188.

[50] ARYA, S. et al. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, v. 45, n. 6, p. 891–923, 1998.

[51] PAREDES, R. et al. Practical construction of k-Nearest Neighbor graphs in metric spaces. In: ÀLVAREZ, C.; SERNA, M. J. (Ed.). *WEA*. [S.l.]: Springer, 2006. (Lecture Notes in Computer Science, v. 4007), p. 85–97.

[52] DONG, W.; CHARIKAR, M.; LI, K. Efficient k-Nearest Neighbor graph construction for generic similarity measures. In: *WWW*. [S.l.]: ACM, 2011. p. 577–586.

[53] AOYAMA, K. et al. Fast similarity search in small-world networks. In: FORTUNATO, S. et al. (Ed.). *CompleNet*. [S.l.: s.n.], 2009. (Studies in Computational Intelligence, v. 207), p. 185–196.

[54] KLEINBERG, J. M. The small-world phenomenon: an algorithmic perspective. In: YAO, F. F.; LUKS, E. M. (Ed.). *ACM STOC*. [S.l.]: ACM, 2000. p. 163–170.

[55] MALKOV, Y. A.; YASHUNIN, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016.

[56] AGGARWAL, C. C.; HINNEBURG, A.; KEIM, D. A. On the surprising behavior of distance metrics in high dimensional spaces. In: BUSSCHE, J. V. den; VIANU, V. (Ed.). *ICDT*. [S.l.]: Springer, 2001. (Lecture Notes in Computer Science, v. 1973), p. 420–434.

[57] COSTA, J. A.; III, A. O. H. Geodesic entropic graphs for dimension and entropy estimation in manifold learning. *IEEE Trans. Signal Process.*, v. 52, n. 8, p. 2210–2221, 2004.

[58] MANDELBROT, B. *Fractal Geometry of Nature*. [S.l.]: W. H. Freeman, 1977.

[59] BRUSKE, J.; SOMMER, G. Intrinsic dimensionality estimation with optimally topology preserving maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, v. 20, n. 5, p. 572–575, 1998.

[60] CAMASTRA, F.; VINCIARELLI, A. Estimating the intrinsic dimension of data with a fractal-based method. *IEEE Trans. Pattern Anal. Mach. Intell.*, v. 24, n. 10, p. 1404–1407, 2002.

[61] AMSALEG, L. et al. Estimating local intrinsic dimensionality. In: CAO, L. et al. (Ed.). *ACM SIGKDD*. [S.l.]: ACM, 2015. p. 29–38.

[62] LIN, K.; JAGADISH, H. V.; FALOUTSOS, C. The TV-Tree: An index structure for high-dimensional data. *VLDB J.*, v. 3, n. 4, p. 517–542, 1994.

[63] BERCHTOLD, S.; KEIM, D. A.; KRIEGEL, H. The X-tree : An index structure for high-dimensional data. In: VIJAYARAMAN, T. M. et al. (Ed.). *VLDB*. [S.l.]: Morgan Kaufmann, 1996. p. 28–39.

[64] CHAKRABARTI, K.; MEHROTRA, S. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In: ABBADI, A. E. et al. (Ed.). *VLDB*. [S.l.]: Morgan Kaufmann, 2000. p. 89–100.

[65] LORENA, A. C. et al. How complex is your classification problem?: A survey on measuring classification complexity. *ACM Comput. Surv.*, v. 52, n. 5, p. 107:1–107:34, 2019.

[66] BRAZDIL, P.; GAMA, J.; HENERY, B. Characterizing the applicability of classification algorithms using meta-level learning. In: BERGADANO, F.; RAEDT, L. D. (Ed.). *ECML*. [S.l.]: Springer, 1994. (Lecture Notes in Computer Science, v. 784), p. 83–102.

[67] THORNTON, C. et al. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: DHILLON, I. S. et al. (Ed.). *ACM SIGKDD*. [S.l.]: ACM, 2013. p. 847–855.

[68] FEURER, M. et al. Efficient and robust automated machine learning. In: CORTES, C. et al. (Ed.). *NIPS*. [S.l.: s.n.], 2015. p. 2962–2970.

[69] BILALLI, B.; ABELLÓ, A.; ALUJA-BANET, T. On the predictive power of meta-features in OpenML. *Int. J. Appl. Math. Comput. Sci.*, v. 27, n. 4, p. 697–712, 2017.

[70] GAMA, J.; BRAZDIL, P. Characterization of classification algorithms. In: PINTO-FERREIRA, C. A.; MAMEDE, N. J. (Ed.). *EPIA*. [S.l.]: Springer, 1995. (Lecture Notes in Computer Science, v. 990), p. 189–200.

[71] CAI, X.; SOWMYA, A.; TRINDER, J. Learning parameter tuning for object extraction. In: NARAYANAN, P. J.; NAYAR, S. K.; SHUM, H. (Ed.). *ACCV*. [S.l.]: Springer, 2006. (Lecture Notes in Computer Science, v. 3851), p. 868–877.

[72] BENSUSAN, H.; GIRAUD-CARRIER, C. G.; KENNEDY, C. J. A higher-order approach to meta-learning. CEUR-WS.org, v. 35, 2000.

[73] LEITE, R.; BRAZDIL, P. Predicting relative performance of classifiers from samples. In: RAEDT, L. D.; WROBEL, S. (Ed.). *ICML*. [S.l.]: ACM, 2005. (ACM International Conference Proceeding Series, v. 119), p. 497–503.

[74] KALOUSIS, A.; HILARIO, M. Model selection via meta-learning: A comparative study. *Int. J. Artif. Intell. Tools*, v. 10, n. 4, p. 525–554, 2001.

[75] SMITH-MILES, K. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, v. 41, n. 1, p. 6:1–6:25, 2008.

[76] GOMES, T. A. F. et al. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, v. 75, n. 1, p. 3–13, 2012.

[77] MORAIS, G.; PRATI, R. C. Complex network measures for data set characterization. IEEE Computer Society, p. 12–18, 2013.

[78] CAMPOS, G. F. C.; BARBON, S.; MANTOVANI, R. G. A meta-learning approach for recommendation of image segmentation algorithms. In: *SIBGRAPI*. [S.l.]: IEEE Computer Society, 2016. p. 370–377.

[79] SMITH-MILES, K. A. Towards insightful algorithm selection for optimisation using meta-learning concepts. In: *IJCNN, part of the IEEE WCCI*. [S.l.]: IEEE, 2008. p. 4118–4124.

[80] PRIYA, R. et al. Using genetic algorithms to improve prediction of execution times of ML tasks. Springer, v. 7208, p. 196–207, 2012.

[81] LEMKE, C.; BUDKA, M.; GABRYS, B. Meta-learning: a survey of trends and technologies. *Artif. Intell. Rev.*, v. 44, n. 1, p. 117–130, 2015.

[82] SOARES, C.; BRAZDIL, P.; KUBA, P. A meta-learning method to select the kernel width in support vector regression. *Mach. Learn.*, v. 54, n. 3, p. 195–209, 2004.

[83] CAMPOS, G. F. C. et al. Machine learning hyperparameter selection for contrast limited adaptive histogram equalization. *EURASIP*, v. 2019, p. 59, 2019.

[84] SUN, Q.; PFAHRINGER, B. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Mach. Learn.*, v. 93, n. 1, p. 141–161, 2013.

[85] YANG, C. et al. OBOE: collaborative filtering for automl model selection. In: TEREDESAI, A. et al. (Ed.). *ACM SIGKDD*. [S.l.]: ACM, 2019. p. 1173–1183.

[86] GUERRA, S. B.; PRUDÊNCIO, R. B. C.; LUDERMIR, T. B. Predicting the performance of learning algorithms using support vector machines as meta-regressors. In: KURKOVÁ, V.; NERUDA, R.; KOUTNÍK, J. (Ed.). *ICANN*. [S.l.]: Springer, 2008. (Lecture Notes in Computer Science, v. 5163), p. 523–532.

[87] KOKIOPOULOU, E. et al. Fast task-aware architecture inference. *CoRR*, abs/1902.05781, 2019.

[88] LI, W. et al. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0). *CoRR*, abs/1610.02455, 2016.

[89] POST, M. J.; PUTTEN, P. van der; RIJN, J. N. van. Does feature selection improve classification? A large scale experiment in OpenML. In: BOSTRÖM, H. et al. (Ed.). *IDA*. [S.l.: s.n.], 2016. (Lecture Notes in Computer Science, v. 9897), p. 158–170.

[90] ZHOU, X. et al. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering*, p. 1–1, 2020.

[91] KRASKA, T. et al. SageDB: A learned database system. In: *CIDR*. [S.l.]: www.cidrdb.org, 2019.

[92] KRASKA, T. et al. The case for learned index structures. In: DAS, G.; JERMAINE, C. M.; BERNSTEIN, P. A. (Ed.). *SIGMOD*. [S.l.]: ACM, 2018. p. 489–504.

[93] AKEN, D. V. et al. Automatic database management system tuning through large-scale machine learning. In: SALIHOGLU, S. et al. (Ed.). *ACM SIGMOD*. [S.l.]: ACM, 2017. p. 1009–1024.

[94] ZHANG, J. et al. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: BONCZ, P. A. et al. (Ed.). *SIGMOD*. [S.l.]: ACM, 2019. p. 415–432.

[95] LI, G. et al. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.*, v. 12, n. 12, p. 2118–2130, 2019.

[96] HILPRECHT, B.; BINNIG, C.; RÖHM, U. Learning a partitioning advisor for cloud databases. In: MAIER, D. et al. (Ed.). *SIGMOD*. [S.l.]: ACM, 2020. p. 143–157.

[97] BARANCHUK, D.; BABENKO, A. Towards similarity graphs constructed by deep reinforcement learning. *CoRR*, abs/1911.12122, 2019.

[98] LI, C. et al. Improving approximate nearest neighbor search through learned adaptive early termination. In: MAIER, D. et al. (Ed.). *SIGMOD*. [S.l.]: ACM, 2020. p. 2539–2554.

[99] RIVOLLI, A. et al. Towards reproducible empirical research in meta-learning. *CoRR*, abs/1808.10406, 2018.

[100] FILHO, R. F. S. et al. Similarity search without tears: The OMNI family of all-purpose access methods. IEEE Computer Society, p. 623–630, 2001.

[101] BEYGELZIMER, A.; KAKADE, S. M.; LANGFORD, J. Cover trees for nearest neighbor. In: COHEN, W. W.; MOORE, A. W. (Ed.). *ICML*. [S.l.]: ACM, 2006. (ACM International Conference Proceeding Series, v. 148), p. 97–104.

[102] BOYTSOV, L.; NAIDAN, B. Engineering efficient and effective non-metric space library. In: *SISAP*. [S.l.]: Springer, 2013. (Lecture Notes in Computer Science, v. 8199), p. 280–293.

[103] HUTTER, F. et al. Algorithm runtime prediction: Methods & evaluation. In: . [S.l.: s.n.], 2014. v. 206, p. 79–111.

[104] EGGENSPERGER, K. et al. Efficient benchmarking of algorithm configurators via model-based surrogates. *Mach. Learn.*, v. 107, n. 1, p. 15–41, 2018.

[105] BREIMAN, L. Random forests. *Mach. Learn.*, v. 45, n. 1, p. 5–32, 2001.

[106] ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: SIMOUDIS, E.; HAN, J.; FAYYAD, U. M. (Ed.). *KDD*. [S.l.]: AAAI Press, 1996. p. 226–231.

[107] CAMPELLO, R. J. G. B.; MOULAVI, D.; SANDER, J. Density-based clustering based on hierarchical density estimates. In: PEI, J. et al. (Ed.). *PAKDD*. [S.l.]: Springer, 2013. (Lecture Notes in Computer Science, v. 7819), p. 160–172.

[108] BARBER, C. B.; DOBKIN, D. P.; HUHDANPAA, H. The Quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, v. 22, n. 4, p. 469–483, 1996.

PUBLICATIONS

Works published by the author during the Master's Degree:

1. Oyamada, R.S.; Shimomura, L.C.; Barbon, S.; Kaster, D.S., **Towards Proximity Graph Auto-Configuration: an Approach Based on Meta-learning**, ADBIS – 24th European Conference on Advances in Databases and Information Systems, 08/2020, Lecture Notes in Computer Science, vol 12245, pp 93-107, DOI: doi.org/10.1007/978-3-030-54832-2_9. (Qualis CC 2016, B1)

2. Shimomura, L.C.; Oyamada, R.S.; Vieira, M.R.; Kaster, D.S., **A survey on graph-based methods for similarity searches in metric spaces**, Information Systems, 02/2020, vol 95, DOI: doi.org/10.1016/j.is.2020.101507. (Qualis CC 2016, A2)