Mining and Querying Ranked Entities

PhD thesis approved by
the Department of Computer science
Technische Universität Kaiserslautern
for the award of the doctoral degree

Doctor of Engineering (Dr.-Ing.)

to

Koninika Pal

Date of defense:
September 11, 2018

Dean:
Prof. Dr.-Ing. Stefan Deßloch

Reviewers:
Prof. Dr.-Ing. Sebastian Michel (TU Kaiserslautern)
Prof. Dr.-Ing. Gerhard Weikum (MPII, Saarbrücken)

PhD committee chair:
Prof. Dr. Heike Leitte

D 386

Abstract

Tables or ranked lists summarize facts about a group of entities in a concise and structured fashion. They are found in all kind of domains and easily comprehensible by humans. Some globally prominent examples of such rankings are the tallest buildings in the World, the richest people in Germany, or most powerful cars. The availability of vast amounts of tables or rankings from open domain allows different ways to explore data. Computing similarity between ranked lists, in order to find those lists where entities are presented in a similar order, carries important analytical insights. This thesis presents a novel query-driven Locality Sensitive Hashing (LSH) method, in order to efficiently find similar top-k rankings for a given input ranking. Experiments show that the proposed method provides a far better performance than inverted-index–based approaches, in particular, it is able to outperform the popular prefix-filtering method. Additionally, an LSH-based probabilistic pruning approach is proposed that optimizes the space utilization of inverted indices, while still maintaining a user-provided recall requirement for the results of the similarity search. Further, this thesis addresses the problem of automatically identifying interesting categorical attributes, in order to explore the entity-centric data by organizing them into meaningful categories. Our approach proposes novel statistical measures, beyond known concepts, like information entropy, in order to capture the distribution of data to train a classifier that can predict which categorical attribute will be perceived suitable by humans for data categorization. We further discuss how the information of useful categories can be applied in PANTHEON and PALEO, two data exploration frameworks developed in our group.

Zusammenfassung

Tabellen oder geordnete Listen sind generische und weit verbreitete Mittel, um Informationen über Entitäten in einer prägnanten und strukturierten Form darzustellen. Ranglisten, wie die höchsten Gebäude der Welt, die reichsten Personen Deutschlands oder die leistungsstärksten Autos sind allgegenwärtige Beispiele. Mit der Verfügbarkeit großer Mengen solcher Ranglisten, die Entitäten kategorisiert nach speziellen Eigenschaften und geordnet anhand unterschiedlichster Metriken in Relation zueinander setzen, können interessante Erkenntnisse hergeleitet werden, in dem berechnet wird welche Ranglisten ähnliche Entitäten ähnlich Ordnen – wobei Rangkriterium und Eigenschaften der Entitäten (stark) unterschiedlich sein können. Dieser Berechnung liegt das Problem der Ähnlichkeitssuche zugrunde. In dieser Arbeit wird dafür ein neuartiger anfragegetriebener Ansatz basierend auf der Idee des Lokalitätssensitiven Hashverfahrens (LSH) präsentiert. Experimente zeigen, dass dieser Ansatz eine weitaus bessere Performanz gegenüber herkömmlichen invertierten Indexen besitzt und auch die populäre Prefix-Filtering Methode in den Schatten stellen kann. Basierend auf dem neu entwickelten Ansatz werden nachfolgend verschiedene Möglichkeiten betrachtet die Größe des Indexes zu beschneiden, bei gleichbleibender oder garantierter Resultatsgüte, realisiert durch Anpassung der Anfrageverarbeitung an die durchgeführte Indexreduktion. Im letzten Teil der Arbeit wird das Problem der Erkennung von Attributen zur sinnvollen Kategorisierung von Mengen von Entitäten. Der vorgestellte Ansatz basiert auf der Berechnung neuartiger statistischer Charakteristiken, ähnlich zur bekannten Entropie aus der Informationstheorie, über den Häufigkeitsverteilungen der den Entitäten zugeordneten Attributausprägungen. Ein Klassifikationsverfahren, über automatisch extrahierten Trainingsdaten aus Wikipedia, wird anschließend benutzt, um vollautomatisch entscheiden zu können, ob ein Attribut geeignet oder ungeeignet ist, Entitäten zu kategorisieren. Benutzerstudien zeigen, dass dies in der Tat möglich ist. Letztendlich wird in dieser Arbeit beschrieben wie und wo die zuvor genannten Techniken in zwei in unserer Arbeitsgruppe entwickelten Datenexplorationssystemen, PANTHEON und PALEO, zum Einsatz kommen.

Contents

Chapter 1

Introduction

The rapid advancement of new technologies and devices is allowing a large number of users to continuously publish digital content on the web. According to a study from IBM[1], 2.5 quintillion bytes of data are generated every day and about 90% of these data are created since 2016—mostly in unstructured or semi-structured formats. Exploiting such an enormous amount of web data for harvesting knowledge is clearly not a trivial task, and inevitably opens a wide range of research problems related to extracting, exploring, and learning information. In order to succinctly summarize and make available factual information about real-world objects (aka. entities) and relations among them from web data, Knowledge bases (KBs), such as Yago, DBpedia, and FreeBase are created. For instance, FreeBase[2] alone contains 1.9 billion facts in their repository and Yago2 [HSBW13] contains hundreds of millions of facts about 9.8 million entities. Moreover, millions of web tables present facts from a widely open domain in semi-structured data formats. To render such vast amounts of data accessible, in this thesis we provide efficient and effective methods in order to create and assess meaningful dimensions for data categorization. Such methods are not only applicable on table-style content in databases or the web, but are also key ingredient to generate ranked lists of entities based on entity-centric facts stored in knowledge bases. Organizing entities in ranked lists, based on some measuring properties, is a traditional solution to present a succinct and easy-to-grasp summary about the top performing members of a group of entities. Rankings are used in various applications in different domains. Examples of such rankings include globally prominent ones, like the list of World's wealthiest persons or tallest buildings. Next to generic tables on the web and automatically generating rankings based on KBs, there exist dedicated websites[3] that are publishing top-k entities every day from different domains. By performing similarity over sets of rankings, with a query being itself a ranked list of entities, users can explore data and acquire hidden insights about the query ranking. For

[1] blog.microfocus.com. Retrieved May 11, 2018.
[2] developers.google.com/freebase/
[3] www.nationmaster.com, www.forbes.com

instance, a user gets to know which properties can also be used as criteria (e.g., gas per mile in the case of cars) in order to create roughly similar ranked lists relative to the query or which ranking criteria for a set of entities in a given query have positive correlation among them. Similarity search over ranked lists is also extensively used in experimental studies. For instance, it is very common to repeat an experiment multiple times in genetics or experimental physics to understand the effect of a specific parameter by studying how often the same experiment produces similar kinds of results.

The main goal of the similarity search over ranked lists is to find all similar rankings to a query very efficiently. In order to be efficient, suitable methods have to find ways to overcome the "curse of dimensionality" and to effectively reduce the potentially very large search space during query processing. In order to reduce the search space, different index structures are developed, such as K-d tree [Ben90], X-tree [VBMS96], generally based on the concept of partitioning or clustering a data space. However, using the idea of smart index structures has no additional advantage over a linear scan, where the dimension of the data becomes higher than ten, as shown by Weber et al. [WSB98]. In this thesis, we deal with top-k lists, where the parameter k is not restricted to any such specific dimension.

Moreover, top-k rankings are naturally incomplete, i.e., it does not rank the entire domain of entities, just the top portion. Hence, in this thesis, we consider the generalized Kendall's Tau distance, defined by Fagin et al. [FKS03]. It is an extension of the famous Kendall's Tau distance, used to assess the similarity between a pair of incomplete rankings. However, the generalized Kendall's Tau distance does not hold the metric property of distance function, and therefore, tree structures like M-tree [CPZ97] that exploit metric property to reduce the search space are not applicable in this work. Instead, inverted indices are deemed to be very effective for efficient similarity search over a large data space, such as text documents [ZM06]. In order to retrieve similar rankings for a user- or application-provided input ranking, the natural idea is to find all those rankings that have one or more entities in common, via an inverted index. Such method retrieves a superset of the final results, i.e., it does not miss any valid answer, but it is not very efficient. To deal with this problem, different prefix-filtering methods [QWL$^+$11, WLF12] are proposed that provide an upper bound on the number of accesses to an inverted index during query processing. We made the key observation that the prefix-filtering method accesses the inverted index far more times than the number of accesses required to retrieve the actual results. Thus, the challenge here is to find a strict upper bound on the number of accesses in an inverted index to retrieve all the results, which directly affects the efficiency of the similarity search. Following the definition of Kendall's tau distance, we can use pairs or even triplets of entities to create an inverted index. This helps to improve the precision of the retrieved candidates significantly. But, as a side effect, the index size and the number of index lookups during query processing grow in an exponential rate, depending on the number of entities that have been used together to create the index. Therefore, in order to optimize the

space requirement, the challenge is to prune the index in a way that maintains the quality of the search result.

The aim of the second part of the thesis is to understand which categorical attributes of a specific entity-centric table will be perceived suitable by humans for the task of defining meaningful subsets of the entity list. This problem can be related to the works on data exploration using OLAP [SAM98] or the faceted exploration in RDF data [ODD06]. To discover interesting data within a database, different OLAP operations are proposed in the literature [JGP16, DPD14], where data experts predefine interesting dimensions in data for applying OLAP operations like drill-down, in order to guide users to explore the data. However, as mentioned earlier, in an open domain scenario, it is impractical to recruit experts to specify which attributes are interesting in a table, continuously emerging in web or generated automatically from KB. Therefore, we want to find the interesting dimensions for exploring data presented in a tabular form, in an automated fashion.

In existing works, an interesting dimension is identified by finding exceptional aggregated result over a measuring attribute, grouped by that dimension [SAM98, THY+17]. We want to remove this constraint of having a measuring attribute always associated with a dimension to capture its interestingness. Therefore, our goal is to identify an interesting categorical attribute, in a human perceived sense, by capturing different characteristics only from the distribution of the categorical attribute. Here, the challenge is finding suitable objective measures that can capture general interests or disinterests of humans on a categorical attribute.

Measures, capturing diversity, unexpectedness, or conciseness of data, are generally used as objective measures to define interestingness in a data. In different contexts of data mining, such as itemset mining or clustering, different interestingness measures [GH06] are used, which we found not directly applicable in our context. Additionally, diversity measures for categorical attributes are rarely discussed in literature. Hence, we need to find objective measures that quantify the interestingness of a categorical attribute tailored to our objective.

Lastly, to the best of our knowledge, there is no catalog available that captures which categorical attributes are preferred by humans to categorize a specific table from all kind of domains. Therefore, we aim to collect such catalog (aka. training data) automatically from web tables, in order to avoid human interactions in building our automated framework of identifying interesting categorical attributes.

1.1 Problem Statement

This thesis deals with two different exploration methods for acquiring insights from rankings or web tables. Here, we will specifically explain each of these problems and the final goals that we want to achieve.

Building	City	Country	Height	Year
Burj Khalifa	Dubai	UAE	828m	2009
Shanghai Tower	Shanghai	China	632m	2014
Abraj Al-Bait Clock Tower	Mecca	Saudi Arabia	601m	2011
Ping An Finance Centre	Shenzhen	China	599m	2017
Lotte World Tower	Seoul	South Korea	554.5m	2016
One World Trade Center	NY City	United States	541m	2014
Guangzhou CTFF centre	Guangzhou	China	530m	2016
Tianjin CTFF centre	Tianjin	China	530m	2018

Table 1.1: The World's Tallest Buildings (Wikipedia)

Consider a table collection $\mathcal{T}$, which comprises a set of ranked lists τ_i and a distance function d. Each $\tau_i \in \mathcal{T}$ contains k entities, associated with its rank based on a criterion. At query time, a user provides a query ranking q of size k and a distance threshold $\theta \in [0, 1]$. In similarity search, our objective is to find efficiently all the rankings from $\mathcal{T}$ that lie within the distance threshold θ from the query q. For example, Table 1.1, showing a part of the top-30 tallest buildings in the world. A user could post the ranking of buildings (the column with the name), here sorted by height, as a query and the system would return all rankings that are similar to this ranking. The user could then learn about alternate criteria that bring rankings in a similar order. For this purpose, we investigate smart lookup methods using index structures to achieve efficiency in the similarity search, by reducing many unnecessary distance computations between retrieved false positive candidates from the index and the query q.

However, the size of an index grows proportionally with the total number of rankings and entities contained in the collection $\mathcal{T}$. In order to optimize the space utilization, we investigate probabilistic pruning methods to prune inverted indices in a way that ensures the quality of the search results, in terms of recall requirement ϱ, given by the user.

Consider further that each τ_i contains additional information about categorical attributes associated with the entities listed in τ_i. For example, Table 1.1 contains additional information such as the place (the country or the city) where the buildings are situated or the year of completion of them. These attributes can be used to categorize the list further. Our goal is to train a classifier that can classify which attributes are meaningful to restrict users' focus into a subset of entities contained in a table τ_i, without any prior knowledge about the domain of the entity list. In order to train the classifier, our first objective is to find suitable statistical measures, such as information entropy, that can provide an objective description (aka. features) of categorical attributes. Then, a machine learning algorithm is applied on extracted features from categorical attributes to train a classifier. Such a classifier can be applied to any table irrespective of whether or not ranks are associated with the list of entities in a table.

1.2 Contributions and Publications

Efficient Similarity search

In order to understand the behavior of the query processing in an inverted index for tuning the retrieval performance, in terms of latency and quality, we relate the inverted index to the concept of Locality Sensitive Hashing (LSH) [AI08]. It is another popular method for similarity search based on hash collisions of similar items inside hash buckets.

We propose two LSH hash families for the generalized Kendall's Tau distance. Using these hash families, we develop four query-driven LSH schemes. For each of them, we derive automatic tuning of LSH parameters, based on the user-defined distance threshold. Additionally, a selection strategy of choosing hash functions for the proposed query-driven LSH schemes is presented. The proposed query-driven LSH schemes are compared to a state-of-the-art method, adaptive prefix-filtering [WLF12], along with other baseline approaches. The results of this work have been published in WebDB'14 [PM14] and SSDBM'16 [PM16b].

- Koninika Pal and Sebastian Michel. An LSH index for computing Kendall's tau over top-k lists. In Proceedings of the 17th International Workshop on the web and Databases (WebDB), 2014.

- Koninika Pal and Sebastian Michel. Efficient similarity search across top-k lists under the Kendall's tau distance. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management (SSDBM), 2016.

To addresses the problem of optimizing the space requirement to store the indices for the proposed LSH methods without compromising the quality of the query results, we discuss three different pruning methods and propose two query processing approaches over pruned indices.

A probabilistic analysis of the effect of index pruning on the quality of search result is devised by modifying the collision probability of proposed LSH. Finally, based on the probabilistic analysis, an optimization problem is formalized to determine the optimal pruning factor, where a user-defined recall goal and the cost of query processing for the similarity search are used as constraints. Besides being used for the proposed LSH method under the Kendall's Tau distance for rankings, the quite generic optimization method can be applied to the case of handling similarity search between sets using the Jaccard similarity.

Experiments show that the optimal pruning factor, derived from the optimization problem, assures the user-defined recall requirement. A detailed study among performances of query processing over each type of pruned indices is discussed, in order to find the best way to process the query over the pruned index. This work has been published in WebDB'17 [PM17].

- Koninika Pal and Sebastian Michel. LSH-based probabilistic pruning of inverted indices for sets and ranked lists. In Proceedings of the 20th International Workshop on the web and Databases (WebDB), 2017.

Mining Interesting Categorical Attributes

We present an automated framework to identify interesting categorical attributes to explore the data in a table. For this purpose, we create a training dataset, where the categorical attributes are collected from Wikipedia tables, and a label, "interesting" or "non-interesting", is associated with each attribute, following the proposed hypothesis. It finds which categorical attributes are interesting for a specific entity type, based on presence or absence of tables in web.

Then, we investigated existing objective measures for categorical attributes, like entropy [SW49], unalikeablity [KP07], etc., which are used to extract features from the training data, in order to train a classifier. Finally, the trained classifier is validated by a user study. This work is presented in a short paper in EDBT'16 [PM16a].

- Koninika Pal and Sebastian Michel. A data mining approach to choosing categorical attributes for ranked lists. In Proceedings of the 19th International Conference on Extending Database Technology (EDBT), 2016.

Further, we extend the previous work by developing three novel objective measures of interestingness for categorical attribute, called p-diversity, max-info-gap, and p-peculiarity. Using these measures, as well as the existing ones as features, a classification model is finally trained with a machine learning algorithm ν-SVM [SSWB00]. A detailed study of the performance of the trained classification models over all possible feature combinations are presented based on a user study. Evaluations show the superiority of the proposed measures p-diversity and max-info-gap over existing measures in capturing interestingness of categorical attribute.

The training data, user study, and the proposed model are publicly available under http://dbis.informatik.uni-kl.de/catmining/. This work has been accepted in SSDBM'18 [PM18].

- Koninika Pal and Sebastian Michel. Learning Interesting Categorical Attributes for Refined Data Exploration. In Proceedings of the 30th International Conference on Scientific and Statistical Database Management (SSDBM), 2018.

The trained classification model has been used in two different systems. The system, PANTHEON that creates a ranking database automatically from the facts in the Yago knowledge base, uses the classifier to filter the non-interesting categories during the ranking generation process. The classifier has been also

used in a reverse engineering system for exploring databases, called PALEO, to filter candidate queries that use non-interesting categories as constraints. This demo has been presented at VLDB'16 [PMMP16].

1.3 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 presents an overview of different index structures and distance measures for efficient similarity search over ranked lists. It further discusses the frequently used statistical measures for capturing different characteristics of the categorical data and a popular learning algorithm and the Support Vector Machine (SVM) for classifying data. Chapter 3 provides a survey of existing literatures on index structures for similarity search and meaningful ways to explore and categorize a list of entities in a table. An efficient similarity search over Top-k rankings by using query-sensitive LSH approach is proposed in Chapter 4. To optimize the space requirement for the proposed approach, different index pruning methods and their effects on the quality of the similarity search are discussed in Chapter 5. A complete framework for training a classifier that identifies interesting categorical attributes for further focusing on a subset of an entity list is proposed in Chapter 6. In Chapter 7, we discuss the applicability of the proposed classifier in two different prototypes, PANTHEON and PALEO, that are mainly focused on the exploration of data. Finally, a summary of the complete thesis is presented in Chapter 8.

Chapter 2

Background and Preliminaries

This chapter presents the background of the basic ideas, methods, and algorithms used in this thesis. It discusses briefly distance functions and index structures commonly used in similarity search. The discussion continues further over statistical measures and classification algorithms to present fundamental concepts in the area of data mining related to this thesis.

2.1 Similarity Search

Information retrieval (IR) is a field of study, where the main focus is finding information from unstructured data to satisfy the information need of users. This field of study intrinsically includes different searching methods and measures to determine the quality of searched results. Here, we present a brief overview of 'similarity search' which covers a broad range of different searching methods, such as range queries, near-proximity search, k-nearest-neighbor. The basic idea is searching a large object space based on a specific measure of relatedness, which acts as the comparator between a pair of objects, to determine the similarity between them. In IR, the objects are often collected from the Internet and are unstructured, e.g., web documents, web tables, weblogs, twitter data. These data objects are represented in different formats based on the applications, for example, documents are denoted as vectors of words in web search engines, entities in a web table can be expressed as a list for the table search, etc. A range query finds all the objects, where the comparison value between the query and the object lies within the range mentioned by users. On the other hand, the k-nearest-neighbor (k-NN) concept aims at finding the k most related objects for a given query.

Consider a data collection $\mathcal{S}$, comprising documents s_i and a distance function d as a comparator. Each set $s_i \in \mathcal{S}$ contains a set of words that describes

the domain of the set s_i, represented as D_{s_i}. The global domain of words is $D = \bigcup_{s_i \in \mathcal{S}} D_{\tau_i}$. At query time, a user provides a query document q and a distance threshold $\theta \in [0, 1]$ for limiting the maximum comparator value or distance. Then, the range query returns all documents $s_i \in \mathcal{S}$ that lie within the distance θ from the query. Therefore, the result set $\mathcal{R}$ of the query q can be written as:

$$\mathcal{R} := \{\tau_i \,|\, d(\tau_i, q) \leq \theta, \tau_i \in \mathcal{S}\}$$

In this thesis, we mainly focus on range queries or near-neighbor queries. In the naïve approach, the result objects are found by comparing each object in the collection with the query, which clearly is not efficient for a large data collection. To avoid such linear scanning, various index structures have been proposed to make the query processing more efficient.

2.1.1 Tree-based Index Structures

To deal with a multidimensional range query or a k-NN query, index structures based on space partitioning such as k-d tree [Ben90], R-tree [Gut84] are an efficient and a prominent solution.

The **K-d tree** is a k-dimensional binary tree that represents the data points in k-dimensional space. It splits the data points based on one of its dimension, at each level of the tree, where the median of data points of the corresponding dimension is considered as the value of the splitting node, to make the tree balance. Given a query q with a distance threshold for a specific dimension, the searching method starts at the root node and prunes the subtrees that do not have any intersection with the query range for that specific dimension.

The **R-tree** is another popular index structure to index the multi-dimensional spatial data that cannot be well-presented by point data. This index structure uses the principle of minimum bounding rectangles that represent the minimum enclosing boxes containing a set of data points. The R-tree is a balanced tree like the B-tree, where each node contains at least half of the maximum number of entries that can be fitted into one node. At leaf level, each bounding rectangle represents a single object. At higher level, it aggregates the objects from child nodes. For range queries, starting from the root node, child nodes are visited only if the bounding box at that node intersects the query region.

Unlike the index structure based on spatial access methods, where search space is partitioned by the actual position of the object in multi-dimensional space, the metric tree [Uhl91] was developed based on the concept of partitioning the search space by the relative distance between objects, to the purpose of reducing the number of distance computations at query time. The search space is pruned by exploiting the triangle inequality property of the distance measure. Combining the advantage of metric tree with dynamic spatial access methods, Ciaccia et al. [CPZ97] proposed the **M-tree**. A node in the intermediate level of an M-tree is called routing node O_r that hold information about its distance from its parent node O_p and the covering radius $r(O_r)$. The covering radius of

a routing node guarantees that the distance of all the nodes under its subtree lies within its covering radius. Due to this property, at the time of range query (q, θ), a subtree is pruned without computing the distance of the query with the node if the the condition $|d(O_p, q) - d(O_p, O_r)| > \theta + r(O_r)$ holds.

2.1.2 Inverted Index

Another popular method which is commonly used in IR for document search is the inverted index. An inverted index consists of two components—a *dictionary* $\mathcal{D}$ of *objects* and the corresponding *posting lists* (aka. index lists) that record information about the object's occurrences in the relation for each objects in the dictionary [ZM06]. For example, considering the set of text documents as objects, the words from each document act as dictionary terms and corresponding posting lists hold document identifiers where the terms appear as shown in Figure 2.1.

Document id	Document
1	The magic power.
2	Secret of the power.
3	Power of magic portion.

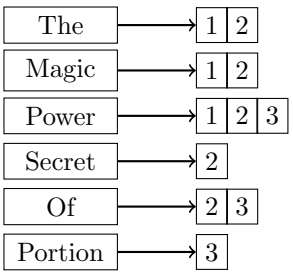


Figure 2.1: Example of inverted index

Depending on applications, posting lists also hold additional informations, such as frequency of the term in the documents, the associated score based on the importance of the term appearing in the documents. A simple **filter and validate** method is used in the query processing to retrieve the results of similarity search as explained in Algorithm 1.

In the filtering step, we access the index for each element from the query to retrieve the candidates. The presence of an element from the query in the index signifies that the objects in the corresponding posting list have an overlap with the query for that very element. Consequently, these objects are considered to be potential candidates to become similar to the query. And finally, candidates are validated by calculating the actual distance to the query.

Algorithm 1: Filter and Validate method

```
1  Procedure filter(Index I, query q)
2      Candidate C ← ⟨ ⟩
3      for element e ∈ q do
4          if e ∈ I.keys then
5              C ∪ {I.postinglist(e)}
6          end if
7      end for
8      return C
9  Procedure validate(C, θ)
10     Resultset R ← ⟨ ⟩
11     for element x ∈ C do
12         distance ← d(x, q)
13         if distance ≤ θ then
14             R ∪ x
15         end if
16     end for
18     return R
```

2.1.3 Locality Sensitive Hashing

Locality sensitive hashing (LSH) is another approach that provides an efficient solution to near-neighbor or c-approximated nearest-neighbor search in high-dimensional data space. The key idea behind LSH is the existence of locality-sensitive hash functions that ensure the property that similar objects have a higher chance to collide (in the same bucket) than the dissimilar ones.

Definition 1 *A Locality Sensitive Hashing scheme is a distribution on a family $\mathcal{H}$ of hash functions, operating on a collection of objects $\mathcal{T} \in \mathbb{R}^d$. With a distance function d, a distance threshold r, and an approximation factor $c > 1$, $h \in \mathcal{H}$ satisfies the following conditions for any two objects $o, q \in \mathcal{T}$:*

- *if $d(o,q) \leq r$ then $Pr_{\mathcal{H}}(h(o) = h(q)) \geq P_1$*
- *if $d(o,q) \geq cr$ then $Pr_{\mathcal{H}}(h(o) = h(q)) \leq P_2$.*

The hash family $\mathcal{H}$ becomes effective for similarity search only when the property, $P_1 > P_2$ is satisfied.

Based on the definition of locality sensitivity, Figure 2.2 shows an example of how a query and objects are distributed over hash buckets according to the distance between them.

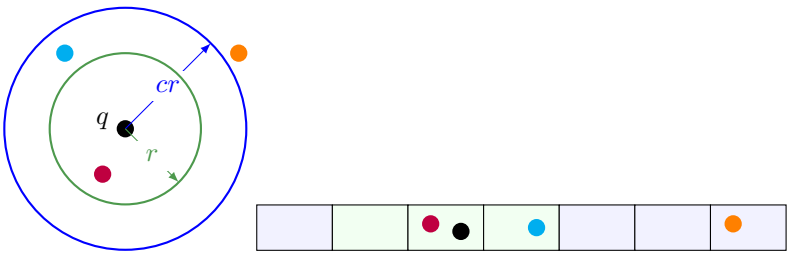


Figure 2.2: Exemplary distribution of points in LSH buckets

Query Processing in LSH Methods

During query processing, a hash function $h \in \mathcal{H}$ is first chosen randomly to map the query q to a hash bucket. All the objects that also are mapped into the same bucket as the query, i.e., $h(o) = h(q)$, are considered potential candidates to be similar with the query. When the gap between the collision probabilities P_1 and P_2 is small, the precision of the candidates becomes low. In order to increase this gap, instead of using single random hash function to map the query into the hash bucket, several random hash functions are concatenated together to create a function family $\mathcal{G}$, which is then used to map the query. The number of concatenated hash functions is parameterized by m. Specifically, with the parameter m, a function $g_j \in \mathcal{G}$ is defined as $g_j = \{h_{1,j}, \ldots, h_{m,j}\}$, where $1 \leq i \leq m$ and $h_{i,j}$ is chosen randomly from $\mathcal{H}$. Clearly, if the more hash functions are concatenated to create g_j, the fewer objects will collied into the same bucket. To deal with this problem, l hash tables are created using $g_1, g_2, \ldots, g_l$ functions to map the query into l hash tables and candidates are collected from all l hash tables to achieve higher recall in query processing. Usually, m is smaller than the dimension of data objects.

Before starting the query processing, LSH requires a preprocessing step in which l hash tables are constructed by mapping each object o from data space to l hash tables by using the function $g_j(o)$ where $g_j \in \mathcal{G}$ and $j = 1, \ldots, l$. Two objects o_1 and o_2 are placed into the same bucket in j^{th} hash table, if $g_j(o_1) = g_j(o_2)$. Then, at query processing, the query q is also hashed to the l hash tables using the same g_j functions which were used at the time of hash table generation. In Algorithm 2, we explain the query processing for the *randomized near-neighbor search problem* using LSH as discussed by Andoni and Indyk [AI08] .

According to Algorithm 2, after mapping the query to the hash buckets, all data points that collide in the same bucket with q in any of the l hash tables are considered as candidates for the query result. Subsequently, the candidates are validated by calculating their distances to the query object.

Probabilistic analysis of LSH: With parameter m, the probability that an object o collides with the query q into the same bucket of j^{th} hash table, for function g_j, i.e., $g_j(o) = g_j(q)$, is at least P_1^m. Therefore, the probability that

Algorithm 2: LSH method for near-neighbor search

```
1  Procedure filter(Index I, query q)
2      Candidate C ← ⟨ ⟩
3      for element e ∈ q do
4          if e ∈ I.keys then
5              C ∪ {I.postinglist(e)}
6          end if
7      end for
8      return C
9  Procedure validate(C, θ)
10     Resultset R ← ⟨ ⟩
11     for element x ∈ C do
12         distance ← d(x, q)
13         if distance ≤ θ then
14             R ∪ x
15         end if
16     end for
18     return R
```

$g_j(o) = g_j(q)$ for *some* $i = 1 \dots l$ is at least $(1 - (1 - P_1^m)^l)$. Hence, the error probability, which is defined by the probability of having a near-neighbor of the query that does *not* collide in any of l hash tables, is at most $(1 - P_1^m)^l$.

With parameters δ (the error probability), m, and l, the probability that a true positive candidates is returned by LSH is given by:

$$1 - \delta \geq 1 - (1 - P_1^m)^l . \tag{2.1}$$

$1 - \delta$ in Equation 2.1 presents the recall function of the LSH method.

In contrast to the LSH, Lv et al. [LJW+07] propose multi-probe LSH that retrieves the candidates from the bucket where the query is mapped and, additionally, probes multiple "close by" buckets to increase the precision of k-nearest-neighbor search. These "close by" buckets are decided by a hash perturbation vector, where the authors restricted the perturbed hash value within $\{-1, 0, 1\}$. According to Figure 2.2, multi-probe LSH collects both the blue and the purple object as candidates at filtering step during query processing, whereas the basic LSH will consider only the purple object as the candidate.

2.1.4 Distance Measures for the Similarity Search

Depending on different application scenarios and the type of objects, different distance functions or similarity measures are used as a comparator in the similarity search. In web search engines, Euclidean distance or Jaccard Coefficient is commonly considered for finding similarities between documents, whereas a distance such as Spearman's footrule distance [Spe94], Kendall's Tau

distance [Ken38], Normalized discounted cumulative gain (NDCG) [JK02], or ERR [Car09] is commonly used for determining similarities between rankings. NDCG or ERR compares two ranked results, retrieved from different searching methods, based on a pre-defined notion of relevance of the retrieved results. Thus, these two measures can not capture the relative binary measure of distance between two ranked lists, consequently, are not suitable for the similarity search problem, addressed in this thesis. In this section, we discuss the Kendall's Tau distance in details with some other distance functions that are considered in this thesis, in order to find the similarity between two rankings or sets.

Kendall's Tau Distance

The Kendall's Tau distance measures the pairwise disagreement between ranking lists. It is calculated over complete rankings that are considered to be permutations over a fixed domain D. A permutation σ is bijection mapping from the domain $D = D_\sigma$ onto the set $[n] = \{1, \ldots, n\}$, where n is the size of the domain $|D|$. For a permutation σ, the value $\sigma(i)$ denotes the rank of the element i. An element i is said to be ahead of another element j in the permutation σ if $\sigma(i) < \sigma(j)$. For two different permutations σ_1 and σ_2 on the same domain D, a pair of elements (i, j) is called *discordant* if i is ahead of j in exactly one of the two permutations, i.e., either in σ_1 or in σ_2, as shown in Figure 2.3 for the pair $(1, 2)$. The Kendall's Tau distance is the total number of discordant pairs between two permutations.

Definition 2 *Given two permutations σ_1 and σ_2 on $D = D_{\sigma_1} = D_{\sigma_2}$, the Kendall's Tau distance $K(\sigma_1, \sigma_2) = \sum_{i,j} \bar{K}_{i,j}(\sigma_1, \sigma_2)$. For a pair $(i, j) \in D \times D$, where $i \neq j$, $\bar{K}_{i,j}(\sigma_1, \sigma_2)$ is defined as follows:*

- $\bar{K}_{i,j}(\sigma_1, \sigma_2) = 0$ *if i and j are in the same order in σ_1 and σ_2.*
- $\bar{K}_{i,j}(\sigma_1, \sigma_2) = 1$ *if they are in reverse order.*

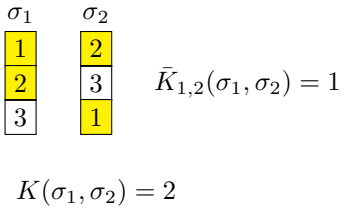


Figure 2.3: Kendall's Tau Distance

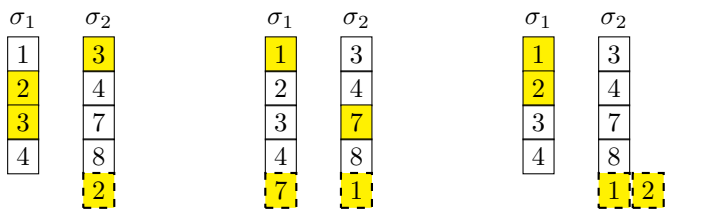
The Kendall's Tau distance holds the metric property and also provides a bound for another popular distance measure, the Spearman's footrule distance, which is defined by the L_1 distance between two permutations.

Generalized Kendall's Tau Distance

In real world scenarios, favorite/preference lists, i.e., top-k rankings are naturally incomplete in nature, capturing only a few top elements of its domain, e.g., top-10 movies or actors of all times. Formally, a top-k list τ is a bijection from D_τ onto $[k]$. The key point is that individual top-k lists, say τ_1 and τ_2, do not necessarily share the same domain, i.e., $D_{\tau_1} \neq D_{\tau_2}$. In this case, we can simply avoid the non-overlapping elements between lists and apply the distance measure on only the overlapping elements. But, in this way, we may lose some useful information for finding the relative comparison between ranked lists. To adapt the incompleteness characteristic of top-k lists in Kendall's Tau and Spearman's footrule distance, Fagin et al. [FKS03] propose the generalized Kendall's Tau and Spearman's footrule distance. The basic idea behind the proposed definition is that all the missing entities in a top-k list are considered to be appearing after the end of the list, i.e., at the $(k+1)^{th}$ position. Based on this assumption, the discordant pairs appear in four different ways, considering the permutation over the complete domain D, i.e., the union of the domain of all top-k lists. For example, with $D = D_{\tau_1} \cup D_{\tau_2}$, Fagin et al. [FKS03] define those four cases as follows:

Definition 3 *Given two top-k lists τ_1 and τ_2 that correspond to two permutations σ_1 and σ_2 on $D_{\tau_1} \cup D_{\tau_2}$, the generalized Kendall's Tau distance with penalty p, denoted as $K^{(p)}(\tau_1, \tau_2) = \sum_{i,j} \bar{K}_{i,j}(\sigma_1, \sigma_2)$ is defined as follows:*

- *Case 1: If $i, j \in D_{\tau_1} \cap D_{\tau_2}$ and their order is the same in both list then $\bar{K}^{(p)}(\tau_1, \tau_2) = 0$ else $\bar{K}^{(p)}(\tau_1, \tau_2) = 1$.*
- *Case 2: If $i, j \in D_{\tau_1}$ and i or $j \in D_{\tau_2}$, let $i \in D_{\tau_2}$ and $\tau_1(i) < \tau_1(j)$ then $\bar{K}^{(p)}(\tau_1, \tau_2) = 0$ otherwise $\bar{K}^{(p)}(\tau_1, \tau_2) = 1$.*
- *Case 3: If $i \in D_{\tau_1}$ and $j \in D_{\tau_2}$ or vice versa then $\bar{K}^{(p)}(\tau_1, \tau_2) = 1$.*
- *Case 4: If $i, j \in D_{\tau_1}$ and $i, j \notin D_{\tau_2}$ or vice versa then $\bar{K}^{(p)}(\tau_1, \tau_2) = p$.*



$\bar{K}_{2,3}(\sigma_1, \sigma_2) = 1$ $\bar{K}_{1,7}(\sigma_1, \sigma_2) = 1$ $\bar{K}_{1,7}(\sigma_1, \sigma_2) = p$

(a) Case 2 (b) Case 3 (c) Case 4

Figure 2.4: Generalized Kendall's Tau Distance

Case 1 of Definition 3 considers the intersection of the incomplete domain of both lists, therefore, it is similar to the definition of the Kendall's Tau distance, presented by Definition 2. The other three cases in the definition of the generalized Kendall's Tau distance consider the complete domain D. Figure 2.4 is showing how the missing elements are placed at the end of the lists according to Definition 3. Unlike other cases, Case 4 in Definition 3 associates a distance penalty p, where $0 \leq p \leq 1$, to a discordant pair if both the elements of a pair are absent from one of the lists. This property gives the distance measure the flexibility to adapt different interpretations of the importance of the missing elements for different applications.

Jaccard Similarity Coefficient

The Jaccard Coefficient [Jac12] is used to determine the similarity between two sets by measuring their mutual overlap. The Jaccard coefficient between set A and B is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

For empty sets, $J(A, B) = 1$. It is a symmetric distance measure, i.e., $J(A, B) = J(B, A)$. For multiple sets, Jaccard coefficient is calculated as follows:

$$J(A_1, A_2, \ldots, A_n) = \frac{|\cap_{i=1}^{n} A_i|}{|\cup_{i=1}^{n} A_i|}.$$

The Jaccard distance which measures dissimilarity between sets is the complement of the Jaccard coefficient. It is given by:

$$d_J(A, B) = 1 - J(A, B).$$

Hamming Distance

Hamming distance [Ham50] is mostly used as an edit distance to find the distance between two strings of equal length in IR systems. It measures the minimum number of substitutions required to convert one string to other. In another word, it counts the number of positions where the symbols between two strings differ. For example, considering a word as an array of letters, 'alice' and 'clare' have different letters in the $1^{st}, 3^{rd}$, and 4^{th} position of the arrays. Hence,

- $d_{hamming}(\underline{a}li\underline{c}e, \underline{c}la\underline{r}e) = 3.$

The hamming distance is extensively used in Information Theory to find the error or distance between bit vectors. In the following example, the left bit vector becomes same as the right bit vector by toggling 4 bits, in the $3^{rd}, 5^{th}, 6^{th}$, and 7^{th} position of the left bit vector. Hence,

- $d_{hamming}(10\underline{0}0\underline{1}0\underline{1}1, 10\underline{1}0\underline{0}\underline{1}\underline{1}0) = 4.$

2.2 Classification of Data

Classification is extensively used in numerous applications, from scientific domains to daily-life business applications, e.g., classifying shape of galaxies, separating normal cells from malignant ones, classifying spam emails. When class labels are associated with training data, a supervised learning method such as decision trees, Bayes classifiers, or support vector machines is used to learn a classification model. On the other hand, when input data contain very few or no information about class labels, semi-supervised or unsupervised learning methods are employed, in order to train the classification model.

2.2.1 Support Vector Machine

Support Vector Machine (SVM) [Vap95] is a well-known supervised learning method for classification, pattern recognition, and regression. The principal idea behind SVM is constructing an optimal marginal hyperplane that separates high-dimensional data into different classes and maximizes the gap between classes, as shown in Figure 2.6(a). Support Vectors (SV) are the data points that lie on the marginal hyperplane. Given the training data which comprises m number of p-dimensional vectors $\mathbf{x_i}$ associated with class labels $y_i = \{-1, +1\}$, the task is to create a binary classifier that can predict whether a vector $\mathbf{x}$ belongs to the class '+1' or '-1'.

There exists a class of hyperplanes defined by $(\mathbf{w} \cdot \mathbf{x}) + b = 0, \mathbf{w} \in \mathbb{R}^p$ and the corresponding decision function (aka. classifier) f is then defined as:

$$f(\mathbf{x}) = sgn((\mathbf{w} \cdot \mathbf{x}) + b), \tag{2.2}$$

where $\mathbf{x} \in +1$ if $f(\mathbf{x}) > 0$, otherwise $\mathbf{x} \in -1$. In the training phase, the parameters $\mathbf{w}$ and b are estimated from the training samples that satisfy the following conditions:

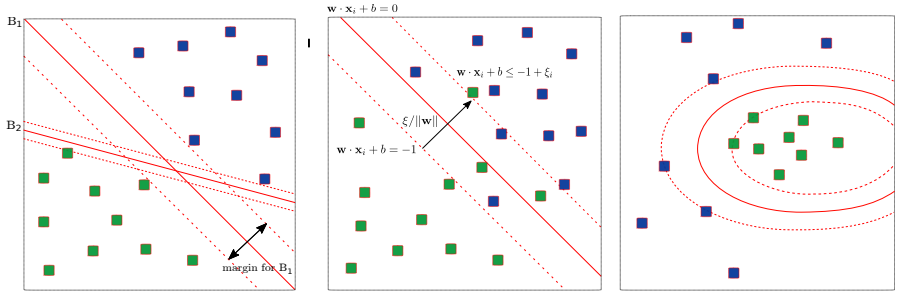
$$\begin{aligned} &\text{given: } (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_m}, y_m), \\ &where \begin{cases} (\mathbf{w} \cdot \mathbf{x_i}) + b \geq 1 & \text{if } y_i = 1 \\ (\mathbf{w} \cdot \mathbf{x_i}) + b \leq 1 & \text{if } y_i = -1. \end{cases} \end{aligned} \tag{2.3}$$

Now, the optimal hyperplane can be found by solving the following optimization problem:

$$\begin{aligned} &\underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}||\mathbf{w}||^2, \\ &\text{subject to} \quad y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1, \quad i = 1, \ldots, m. \end{aligned} \tag{2.4}$$

In general, the dual form of the optimization problem in Equation 2.4 is solved to determine the optimal hyperplane.

In practice, data cannot always be separated by a hyperplane due to the presence of noise in the training data, as shown in Figure 2.6(b). To handle



(a) Decision boundaries (b) non-separable data (c) non-linear data

Figure 2.5: Classifying data with SVM, (from [TSK05])

this limitation of the linear SVM, we can use soft-margin approaches which introduce a positive-valued slack variable ξ in the optimization problem to learn the decision boundary that is tolerable to small training errors. Cortes and Vapnik [CV95] present C-SVM, a soft-margin approach with a regularization parameter $C > 0$ to control the training error. A low C-value allows more support vectors, which lie within wrong class, during learning procedure to make the classification task easy. On the other hand, $C = \infty$ makes the classification problem strict. Replacing this free parameter C with the parameter ν, Schölkopf et al. [SSWB00] propose ν-SVM to create a robust classification model from a noisy training data. It modifies the optimization problem in Equation 2.4 as follows:

$$\begin{aligned} &\underset{\mathbf{w},\xi\in\mathbb{R}^p,\nu,b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{2}||\mathbf{w}||^2 - \nu\rho + \frac{1}{2}\sum_{i=1}^{m}\xi_i, \\ &\text{subject to} \quad y_i \times ((\mathbf{w}\cdot\mathbf{x}_i) + b) \leq \rho - \xi_i, \quad i = 1, \ldots m \\ &\quad \text{and} \qquad \xi_i \geq 0, \quad \rho \geq 0. \end{aligned} \tag{2.5}$$

Here, training points with $\xi_i > 0$ represent marginal errors, i.e., the training points that lie either within the margin or the wrong side of the boundary. Parameter ν lies in $[0, 1]$. It provides the lower bound on the fraction of support vectors and an upper bound on the fraction of the marginal errors, i.e., the number of points that are misclassified or lie within the margin. As ν increases, the classifier allows more points to lie within the margins, and therefore, the model can be underfitting the data. The optimization problem in Equation 2.5 is solved by the dual problem given in Equation 2.6 with the dual variable $\boldsymbol{\alpha}$;

details can be found in [SSWB00]:

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}}\ \mathbf{W}(\boldsymbol{\alpha}) &= \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j(\mathbf{x_i}\cdot\mathbf{x_j}), \\ \text{subject to}\quad 0 \le \alpha_i &\le \frac{1}{m}, \quad \forall i \\ \sum_{i=1}^{m}\alpha_i y_i = 0, &\quad \sum_{i=1}^{m}\alpha_i \ge \nu. \end{aligned} \tag{2.6}$$

The decision function in Equation 2.2 then becomes:

$$f(\mathbf{x}) = sgn(\sum_i \alpha_i y_i(\mathbf{x}\cdot\mathbf{x_i}) + b). \tag{2.7}$$

In SVM, the similarity between data is expressed in the *inner product space* as it is a linear classifier. However, it is possible that the data may have non-linear decision boundaries, as shown in Figure 2.6(c). In such cases, SVM can employ a non-linear mapping that transforms the data $\mathbf{x}$ from original data space to a higher dimensional space $\Phi(\mathbf{x})$ such that a linear decision boundary can separate the data in the higher dimensional space. To implement the non-linear classifier, we generally use kernel trick that measures the similarity between data points in higher dimensional space using the original data space. Different kernel methods are used in SVM to find the similarity in the feature space as discussed in literature [SS01]. The radial basis function (RBF) is one of the popular kernel methods, commonly used for non-linear SVMs [CHC+10]. It is given by:

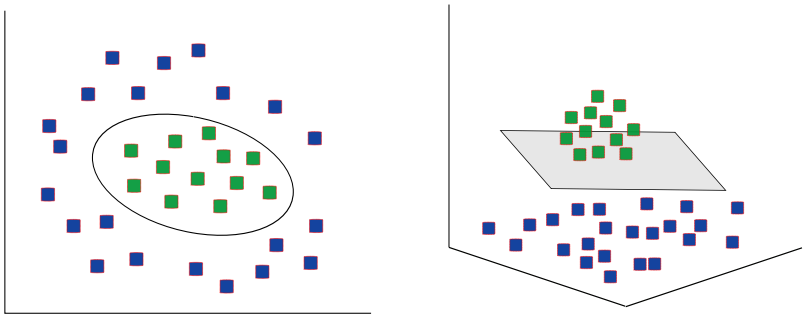
$$(\Phi(\mathbf{x})\cdot\Phi(\mathbf{x}')) = K(\mathbf{x},\mathbf{x}') = exp(-\gamma\|\mathbf{x}-\mathbf{x}'\|^2), \quad \gamma > 0.$$

Here, γ is the free parameter in the RBF kernel, where it determines the influence of a single support vector on decision making. A large γ-value signifies small Gaussian variance, which implies that the radius of the area of influence of a support vector only includes the support vector itself. Hence, the resulting model can be overfitting the data. On the other hand, a very small γ-value implies that the radius of the area of influence of a support vector is very large and the resulting model can fail to capture the complexity of the data.

In this thesis, we use LIBSVM [CL11], a very popular library which provides implementations for different variations of SVM. It allows to plug in different kernel functions to SVM and provides a parameter-selection mechanism for C-SVM. The library LIBSVM is used as back-end of many classification or regression tools available, such as Weka [HFH+09].

2.2.2 Statistical Measures

Raw data are often expressed as records, vectors, lists of entities, etc., which may need to be preprocessed to represent the data in a form that captures the



(a) non-linear decision boundary in the original data space

(b) linear decision boundary in the higher dimensional space

Figure 2.6: Kernel trick for non-linear SVM[1]

characteristics of the data and can be accepted as an input to a classification algorithm. Depending on the input data types, different statistical measures can be used to capture the fundamental characteristics of the data. As we are dealing with the classification of categorical data in this thesis, we discuss here commonly used statistical measures for categorical data.

Entropy

One of the popular measures, used extensively in classification of categorical data is information entropy. It was first introduced in the mathematical theory of communication [SW49]. It is defined to measure the uncertainty of the outcome of a discrete information source. Let us consider a random variable X, where $I(X)$ represents the information content of it. $I(X)$ is defined by $-\log_b(P(X))$, where $P(X)$ presents the probability mass function of the random variable X. Then, the entropy $H(X)$ is defined by the expected value of $I(X)$.

$$H(X) = E[I(X)] = -\sum_{x_i \in X} P(x_i) \log_b P(x_i)$$

Shanon entropy considers 'bits' as the unit of entropy, where the base b is set to '2'. In data communication, entropy is used to find average length of compressed message. Binary classifier uses entropy as an impurity measure. In this thesis, we use entropy to express the average information content captured by an attribute, where $P(x_i)$ is nothing but the frequency of a value x_i of the attribute.

Max-Coverage

For a categorical attribute, frequency is a basic building block to define different measures to characterize the distribution of categorical values. Consider a

[1] https://www.hackerearth.com/blog/wp-content/uploads/2017/02/kernel.png

categorical attribute X that can take n categorical values $\{x_1, x_2, \ldots x_n\}$ for a set of N objects. Then, the frequency of a categorical value x_i is calculated as:

$$freq(x_i) = \frac{\#(\text{objects that hold attribute value } x_i)}{N}.$$

$freq(x_i)$ is equivalent to the coverage of x_i on a set of given objects. It is frequently used in itemset mining to capture how often a item appears in a set of transactions. It is also used to calculate mode of a categorical attribute, i.e., the categorical value which has highest frequency. For a categorical attribute with few categorical values, mode and frequency are considered important measures for the exploration of data. Rather using mode, in this thesis, we use Max-Coverage, which is the frequency of mode, given by:

$$mCov(X) = \max_{x_i \in X} freq(x_i).$$

Unalikeability

Variance is a common measure for describing the *degree of diversity*. Unlike numeric data, the distance between two observations is always boolean for categorical data. If both observations hold the same categorical value then the distance is 1 otherwise 0. Following this property, Kader and Perry [KP07] discuss a variation coefficient for categorical data, called unalikeability, denoted by $U(X)$ for a random variable X. Instead of measuring how much an observation of random variable differs, it measures how often observations of a random variable differ from one another. For a random variable X, unalikeability is calculated as:

$$U(X) = 1 - \sum_{x_i \in X} freq(x_i)^2.$$

The value of Unalikeability lies within [0,1]. Higher value of unalikeability signifies more diverse data.

Peculiarity

Another popular statistical measure that is discussed in various fields, such as mathematical ecology, microbiology, or economics, is the Simpson index. It captures the degree of concentration of categorical values which are associated with a set of objects. It is defined by the probability of two objects being associated with the same categorical value, where objects choose categorical values randomly without replacement. The collection diversity is defined as the complement of Simpson index. Let us consider a random variable X that can take n categorical values, $\{x_1, x_2, \ldots x_n\}$ for a set of N objects. Then, the collection diversity, referred as peculiarity measure in this thesis, denoted as $D(X)$, is calculated as:

$$D(X) = 1 - \sum_{x_i \in X} \frac{count(x_i)(count(x_i) - 1)}{N(N-1)},$$

where $count(x_i) = \#$(objects that hold attribute value x_i). When data is very large, $D(X)$ becomes approximately identical to $U(X)$, because the probability of two objects that choose the same categorical value without replacement or with replacement becomes almost equal.

2.3 Evaluation Measures

The efficiency of both retrieval and classification algorithms can be measured easily by computing the response time for queries or space complexity of algorithms. Nevertheless, an efficient method that produces useless results does not satisfy the information need of users. Therefore, one of the main concerns in both retrieval and classification algorithms is to measure the effectiveness of a method, based on the quality of the produced results. The quality of IR-based systems or classification models is generally quantified by the notion of user utility and information need. Now, evaluating effectiveness can be critical as the perception of the quality may differ among users.

The standard approach to measure the effectiveness of IR-based systems is performing relevance assessment of the produced results. For the relevance assessment, a ground truth for a query is created by marking each document either relevant or non-relevant to the query in advance, according to the information need of users. Similarly, the ground truth for the classification task holds the true class label for the test samples. Given the ground truth, precision and recall are the most commonly used measures to evaluate the quality of produced results in both areas.

Precision

In IR-based systems, when a query returns a set of documents, precision is calculated by the fraction of retrieved documents that are relevant.

$$\text{Precision} = \frac{\#(\text{relevant documents retrieved})}{\#(\text{retrieved documents})}$$

In classification tasks, for a specific class c_i, the precision is the fraction correct predictions over all the objects that are predicted to be in c_i. Precision is considered as class-specific accuracy in classification task.

$$\text{Precision} = \frac{\#(\text{objects predicted to be in the true class } c_i)}{\#(\text{objects predicted to be in class } c_i)}$$

Recall

Recall is the fraction of relevant documents that are retrieved.

$$\text{Recall} = \frac{\#(\text{retrieved relevant documents})}{\#(\text{relevant documents})}$$

For a specific class c_i, recall is the fraction of correct predictions over all the objects in class c_i according to the ground truth. Recall is also called class-specific coverage in classification.

$$\text{Recall} = \frac{\#(\text{objects predicted to be in the true class } c_i)}{\#(\text{ objects in class } c_i \text{as per ground truth})}$$

Accuracy

Accuracy is the fraction of objects that are correctly predicted its true class over all objects.

$$\text{Accuracy} = \frac{\sum_{c_i} \#(\text{ objects predicted to be in the true class } c_i)}{\#(\text{All objects})}$$

In another word, accuracy is the overall precision. Classification error is the complement of classification accuracy.

In IR-based system, accuracy is calculated as:

$$\text{Accuracy} = \frac{\#(\text{retrieved rel. doc.} + \text{ non-retrieved non-rel. doc.})}{\#(\text{All documents})}.$$

Accuracy is considered a less informative measure in both areas. Specifically, in IR system, it is not considered as an appropriate effectiveness measure. Consider a scenario, where very few documents are relevant. In this case, a searching algorithm can reach a high accuracy by simply not retrieving any document.

F-measure

Depending on applications, a tradeoff between precision and recall is necessary. We can retrieve all the documents or classify all the object to a specific class to achieve 100% recall. Definitely, the precision will suffer in this scenario. On the other hand, we can develop an algorithm that retrieves only highly relevant documents to achieve high precision which will affect the recall of the system. F-measure is a metric that tradeoff between precision and recall. It is nothing but the weighted harmonic mean of precision and recall, defined as follows:

$$F_\beta = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}, \quad \text{where } \beta = \frac{1-\alpha}{\alpha}.$$

Here, P and R represents precision and recall respectively, $\alpha \in [0, 1]$, and $\beta^2 \in [0, \infty]$. Hence, F_1 measure balances precision and recall equally.

Fleiss Kappa

The ground truth, which is used to measure the quality of a IR-system or a classification method, must be constructed by domain experts to reflect the

Kappa	Agreement
<0	Less than chance agreement
0.01 - 0.20	Poor agreement
0.01 - 0.20	Slight agreement
0.21 - 0.40	Fair agreement
0.41 - 0.60	Moderate agreement
0.61 - 0.80	Substantial agreement
0.81 - 0.99	Almost perfect agreement

Table 2.1: Interpretation of kappa values, from [VG05]

information needs at its best. After creating the test collection, relevance assessments from humans are put together to create the ground truth. As the perception of relevance may differs among humans, instead of using a single assessor, assessments from many judges are considered as a reliable source to generate the ground truth. In this scenario, it is also important to measure the reliability of agreements in the assessments given by the judges. One such common reliability measure is kappa statistics.

Fleiss' kappa [Fle71] measures the degree of agreement present in judgements compare to the agreements that would be expected by chance. It is computed as:

$$kappa = \frac{P(A) - P(E)}{1 - P(E)},$$

where $P(A)$ represents the fraction of the times when judges are agreed and $P(E)$ represents the fraction of the times the agreements would be expected by chance. The value of *kappa* lies within the range $[1, -1]$, where '1' stands for complete agreement among judges and '0' indicates that the agreements among judges happened by chance. An negative value of *kappa* indicates that the agreements among judges worse than random or cab be biased. Viera and Garrett [VG05] present a subjective interpretation of different ranges of kappa values with the different level of agreements, as shown in Table 2.1.

Chapter 3

Related Work

This chapter presents an overview of the state-of-the-art in the vicinity of the research problems addressed in this thesis. First, in Section 3.1, we review the literature that discuss the applicability of different distance measures in rank similarity and existing approaches for finding similar rankings efficiently. We further continue the discussion of the existing works on pruning of index structures for optimizing the space requirement in similarity search, in Section 3.2. After that, we discuss the existing works on identifying suitable attributes for categorizing a list of entities present in a table. More specifically, we review existing methods for exploring tables, extracting categorical attributes from tables, and quantifying interestingness of categorical data, in Section 3.3.

3.1 Similarity Search over top-k Lists

With the availability of billions of tables, different web search engines, and crowdsourced information in web, the problem of effective and efficient comparison of top-k lists can be associated with different applications and has been addressed by many researchers. Few common application areas are diversifying the results retrieved from a search engine [QYC12, AGHI09], comparing the results retrieved from different search engines [BML06], rank aggregation problem [SvZ09, DKNS01], k-nearest-neighbor search on rankings, etc. Depending on the application scenario and data type, different distance measures are used to find the similarity between rankings, such as Jaccard Distance, Hamming distance [Ham50], Spearman's footrule distance [Spe94], Kendall's Tau distance [Ken38], Pearson correlation. As we are dealing with the problem of similarity search on incomplete rankings in this thesis, we investigate specifically the distance functions that provide correlation measures over incomplete top-k lists [FKS03, BML06, WMZ10], such as generalized footrule or Kendall's Tau distance, rank-biased overlap.

It is difficult to single out one measure that is appropriate for different applications. To compare the applicability of these measures, Kumar et al. [KV10]

discuss criteria that arguably should be fulfilled by a distance measure used for comparining ranked lists. The criteria include properties like richness, simplicity, generalization, scale-freeness, and correlation with other distance measures. With the generalization of the definition of the Spearman's footrule and Kendall's Tau distances, they fail to meet the simplicity criteria. Nevertheless, the generalized Kendall's Tau distance can still bound loosely the generalized footrule distance and an equivalence relationship between them is possible to define [FKS03, KV10]. Another work by Sun et al. [SLC10] discuss five essential properties for a distance measure to be effective for finding rank similarity in the context of web search. According to these properties, a distance measure should be—symmetric, applicable to incomplete list, flexible to adapt user's preference on top or bottom ranks, computationally efficient, and able to aggregate information from multiple queries for finding the final dissimilarity. Generalized Kendall's tau distance can capture all these properties except the third one. Overall, Generalized Kendall's tau distance becomes one of the most suitable distance functions to be used in similarity search over rankings and is considered as distance function in this thesis.

In the next section, we will review the available indexing methods, commonly used for increasing the efficiency of the similarity search.

3.1.1 Index Structures for Similarity Search

In similarity search, retrieval is done based on the query's items; the result is a set of candidate rankings on which the distance function is applied to find the similarity with the query. The main objectives are to avoid full scan through all rankings and minimize the number of the distance computation, in order to find the search results efficiently. In this section, we discuss existing works that propose suitable index by exploiting the characteristics of the distance functions, to find a reduced search space on which the actual comparison with the query is performed to retrieve the results.

While using distance measures that satisfy the metric property, i.e., the triangle-inequality property, we can use data-agnostic structures, like the *metric tree* by Uhlmann [Uhl91], for partitioning the data. Optimizing the distance computations by using properties of metric tree and the I/O access, Ciaccia et al. propose the M-tree [CPZ97]. This tree structure is widely used in similarity search [BFL$^+$10] and clustering method [ZGZG11]. Further, Zezula et al. [ZSAR98] present three algorithms for the k-nearest neighbor search using M-tree by introducing the relative distance error, exploiting the distance distribution from a object's viewpoint, or finding *potentially optimal point* for performance improvement. But, as mentioned earlier, top-k lists are incomplete, and hence, measures like generalized footrule or the Kendall's Tau distance can be used in similarity search over incomplete top-k lists. Fagin et al. [FKS03] prove that generalized footrule distance satisfy *near-metric* property, given that the rank of missing elements must be bigger than k. Milchevski et al. [MAM15] consider the generalized footrule distance in their proposed method for the efficient

similarity search over incomplete rankings. They combine metric space indexing with inverted indices for reducing the search space and the number of distance computations during query processing. However, the generalized Kendall's Tau distance does not hold the metric property. Therefore, metric trees cannot be used to index the data objects.

Besides tree-based index structures, the inverted index provides an effective solution for indexing documents in text-based search engines, like *Google or Yahoo*. Zobel and Moffat [WMZ10] present a detailed study on construction, maintenance, and representation of an inverted index for keywords queries in text search engine. The authors also discuss drawbacks of suffix arrays and signature files [CS88] for document search, which are also applicable for similarity search over ranked lists. Additionally, existing works on set joins [CGK06] or querying set-valued attributes [HM03], using inverted index, are also relevant to discuss here, as top-k lists can be considered as plain sets. Helmer and Moerkotte [HM03] discuss four index structures, sequential signature file, signature tree, extendible signature hashing, and inverted files, which can be used for handling queries on set-valued attributes. Through a comprehensive experimental study, the authors also show that the performance of query execution over an inverted file outperforms the other index structures. To optimize the performance of set similarity join, the concept of prefix-filtering approach has been first introduced for different edit distances and overlap-based distance functions, by exploiting the similarity threshold [SK04, CGK06].

Similarity search using an inverted index follows a simple filter and validate methods as shown in Algorithm 1. Exploiting the user given or system specified similarity threshold, we can apply the concept of the prefix-filtering method that finds a bound on the number of index scans during query processing, and thus, can prune the search space in the first place, for the similarity search. Xiao et al. [XWL$^+$11] propose a position-based prefix-filtering method for near-duplicate search, where they adapt the prefix-filter parameter for Jaccard similarity, hamming distance, edit distance, and cosine similarity. In this thesis, we discuss prefix-filtering parameters for generalized Kendall's tau distance and Jaccard similarity, which can also be seen as a position based a prefix-filtering method. Qin et al. [QWL$^+$11] present a signature-based filtering method for the edit-similarity join problem, where they show that the lower bound for the signature size is $\tau + 1$, where τ is the edit distance threshold. Wang et al. [WLF12] propose an adaptive prefix-filtering framework for similarity joins, called SimJoin. They sorted the elements in each set according to a global ordering of all elements, in order to generate the prefix-elements for each set. Considering a overlapping threshold t, prefix elements of a set r are the first $|r| - t + 1$ elements of r, called the 1-prefix scheme. Generalizing the prefix scheme, they discuss the l-prefix scheme that create prefix elements of a set of r by using the first $|r| - t + l$ elements of r. They observed that the parameter l has a significant effect on the performance in set similarity join, as shown in Figure 3.1. They propose a cost model for variable-length prefix scheme and a greedy algorithm to find optimal l-prefix scheme for a given query, based on

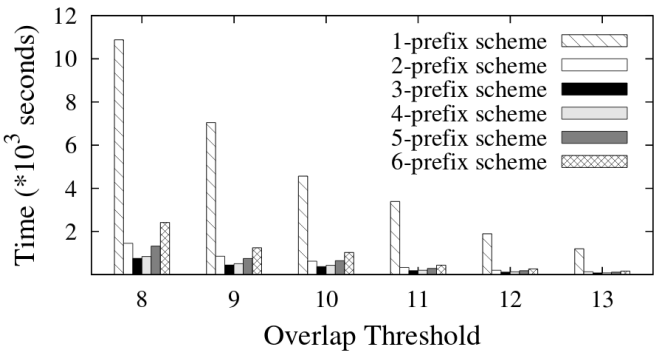


Figure 3.1: Performance of different prefix-filtering schemes for overlap similarity between sets, from [WLF12]

their observation. For adapting the variable-length prefix scheme, they also propose an incremental index structure, called the delta inverted index. According to this work, the basic prefix-filtering method is a specific case of the proposed adaptive prefix-filtering approach, where the tuning parameter l of the adaptive prefix-filtering is set to 1. They found that often $l \geq 1$ performs more efficiently compare to basic prefix-filtering method. In this thesis, we use the adaptive prefix-filtering method as a competitor to our proposed approach.

Continuing the discussion of existing approaches for efficient similarity search, we review existing works on the another popular approach, Locality Sensitive Hashing, in the next section.

3.1.2 Locality Sensitive Hashing

Wang et al. [WSSJ14] provide a detailed survey of mainly two kinds of hashing methods, locality sensitive hashing (LSH) and learning to hash, applied in many scenarios, such as similarity search, clustering. With the aim of reducing the cost of similarity search, approximate similarity search, where the correctness of the candidate set is relaxed with some error bound, is one of the solutions provided by LSH [AI08, DWJ+08, DIIM04, GIM99]. Indyk and Motwani [IM98] first introduce the concept of LSH for similarity search over high-dimensional data. The key idea behind LSH is the use of locality-preserving hash functions that map close objects to the same hash value (i.e., hash bucket) with high probability, discussed in Section 2.1.3. The authors propose LSH families for hamming distance and l_p-norm, where $p \in [1, 2]$. They theoretically establish that the time complexity is bounded by the exponential factor of $1/c$ over the data points for c-approximate nearest-neighbor problem. Gionis et al. [GIM99] show that the proposed LSH method in [IM98] significantly improves the performance of similarity search and outperforms SR-tree that is proposed by Katayama and Satoh [KS98] to handle the near-neighbor search in high-dimensional data. Datar et al. [DIIM04] propose a hash family for l_p

distances in p-stable distributions, where $p \in (0, 2]$. For the Euclidean distance, the authors also show that the exponential factor in time complexity of c-approximate near-neighbor problem for the proposed LSH family is strictly less than $1/c$. To detect near-duplicate web pages, Broder [Bro97] proposes an LSH family, min-hash that approximates the Jaccard similarity between two sets. For the angular distance between two vectors, Charikar [Cha02] defines an LSH family that randomly chooses a unit-length vector from d-dimensional space. Then, the chosen unit-length vector is used to divide the space into two half spaces and projects any vector into one of these half spaces.

Different parameters of locality-preserving functions render LSH a parametric approach. Dong et al. [DWJ+08] present a performance model of multi-probe LSH on gamma distributions, which tunes different LSH parameters, such as l and m, to reach optimal average performance. It also determines the number of bucket to probe in a hash table adaptively in query time. Joly and Buisson [JB08] propose another variant of multi-probe LSH where a posteriori model is first learned from prior queries and results. Based on the learned model, it detects which buckets to probe. Instead of using compound hash functions or probing multiple buckets for one collision, Gan et al. [GFFN12] propose Collision Counting LSH to ensure high recall for the searching method. The proposed method choose a function base of some single hash functions from the hash family and a collision threshold in a way such that if an object collides with query more than the threshold times, the object becomes a good candidate. Bawa et al. [BCG05] propose the LSH Forest indexing method where the labels in each hash table are presented by a prefix tree. According to different queries, it determines the number of hash functions m it needs to access for each table, in order to guarantee a certain level of search result quality. Satuluri and Parthasarathy [SP12] propose the BayesLSH method based on Bayesian statistics. It prunes the data points during candidate generation by inferring the probability that the similarity between two data points lies above the given distance threshold by comparing fewer hash functions. The authors discuss BayesLSH approach for Jaccard and cosine similarity in all-pair similarity problem. Liu et al. [LCH+14] propose sorting-key LSH, a smart method to access the page that holds candidate objects on the disk. They calculate the distance between two hash keys in a fashion that reflects the distance between two objects; the closer the hash keys smaller the distance between objects. Therefore, they keep the objects from close hash buckets locally in an index file, which helps to reduce significantly the number of disk accesses to generate enough candidates that ensure the result quality. Huang et al. [HFZ+15] propose a query-aware LSH scheme for the hash family proposed in [DIIM04], where the partition of the hash bucket is done based on the point where the query is mapped.

All the LSH families discussed above are defined for distance measures in metric space. Athitsos et al. [APPK08] propose the distance-based hashing scheme that is also applicable for non-metric distance measures. They propose a hash family that projects a data point in binary space, based on the "line projection" function, defined in the paper. The authors also present a cost

model and an incremental way to find the LSH parameter k and l. Jégou et al. [JASG08] present a query-adaptive LSH method where only the highest relevant hash functions to the query are selected based on the E_8 lattice formation of them, to solve the nearest-neighbor problem. Gao et al. [GJLO14] present data-sensitive hashing for c-approximate k-Nearest-Neighbor (k-NN) search. Their proposed hash family is the trained binary linear classifiers that can classify the k-NN data points from non-ck-NN ones. In this thesis, we are also working with a non-metric distance function, the generalized Kendall's Tau distance, and propose two hash families that project the data points in binary space.

3.2 Index Pruning

The space requirement an inverted index is directly proportionate to the vocabulary of the document collections or the number of objects in the data space. Additionally, it is possible to index documents using k-shingling, pairs, or triplets instead of single words from documents. This increases the precision of similarity search as well as the space requirement to store the index. Similarly, LSH methods maintain many hash tables to ensure a good result quality in the approximate similarity search. Clearly, the space requirement increases due to multiple replications of data objects in both the cases. As a consequence, a compromise between the result quality and the space requirement comes to the scenario. In this section, we briefly discuss existing works that address this problem.

3.2.1 Optimizing Space Requirement in LSH

The previous section describes many variants of locality-sensitive hash families for different metric, non-metric, and Euclidean distance functions. One principal drawback of the LSH method is maintaining a large number of hash tables to reach a high result quality for the similarity search. To reduce the space requirement, Panigrahy [Pan06] proposes an entropy-based LSH approach. It creates fewer hash tables by randomly choosing data points as additional queries with the original query. Lv et al. [LJW+07] propose the multi-probe LSH method that finds hash buckets near to the buckets where the query is mapped to using a fixed sequence of perturbed vectors, and collects candidates from all of them to reach the same result quality with fewer number of hash tables. Few variations of multi-probe LSH are proposed in [DWJ+08, JB08], discussed in the previous section, which also optimize the space requirement. Tao et al. [TYSK10] propose a locality-sensitive B-tree structure using the Z-order value of the projection of the data objects for compound hash functions, to avoid storing a single object in multiple hash tables. All these proposed LSH approaches focus on optimizing the space requirement for LSH family for p-stable distribution [DIIM04], where Euclidean distance is used as the distance measure. On the other hand, query-aware LSH methods, discussed in [JASG08, HFZ+15] select hash func-

tions during query time, which demands to keep most of the hash tables created from the hash families, though very few of them are actually used at query time. Such inefficient usage of space is also a drawback of our proposed query-driven LSH method discussed in Chapter 5.

3.2.2 Index Pruning in Document Search

To optimize the space requirement in document search engine, different index pruning methods are proposed in literature. Static pruning methods mainly focus on pruning elements from an index in a way that can still capture the top-k results for a query in document search. Carmel et al. [SCC+01] introduce a term-based static index pruning method where index keeps only a few top-scored documents in a posting list corresponding to a term. The authors employ a $tf \times idf$-based scoring function for the documents, where the cut-off score for posting list is term-dependent. Modifying the method in [SCC+01], de Moura et al. [dMdSF+05] propose a locality-based pruning approach by keeping top documents for each term with top documents of its related terms. Büttcher and Clarke [BC06] propose a document-centric index pruning method where few top-scored terms from a document are kept in the index. The score of the terms in a document is assigned by a feedback relevance method based on Kullback-Liebler divergence of the documents in the collection. Nguyen [Ngu09] proposes a posting-based pruning method which keeps only a few top-scored postings. The score of a posting is calculated by combining the assigned weight of the document and the term in the posting. There exist other static pruning methods that consider additionally the distribution of query data in a term- or document-centric pruning method [AOU12, JRS16]. These approaches are fully orthogonal to our index pruning method and can be easily adopted in the proposed optimization problem for pruning the LSH index, discussed in Chapter 5. On the other hand, dynamic index pruning methods focus on reducing query processing time by eliminating redundant operations and maintaining a cache [BCH+03, AM06, TTZ07]. In this thesis, the access method of the inverted index is derived from LSH, and hence, we leave out here the detail discussion of dynamic index pruning.

3.3 Entity-centric Category Mining

The data we are dealing with in this thesis are rankings and tables in web, capturing many entities and their properties in a semi-structured manner. Understanding and exploring such data is a primary necessity for scientific discovery. In this section, we will review existing methods, frequently used by the database community, for handling these two tasks. Additionally, the area of data mining provides a large set of algorithms to discover hidden characteristics and patterns of the data, used in different application scenarios to meet user requirement [TSK05]. As rankings or web tables are very dynamic in nature, in

this section, we will briefly cover the adaptation of data-driven approaches for exploring data, in both of the database management and the data mining area.

In databases, the stored data normally less dynamic in nature, i.e., the data do not change very often, allowing data analysts use OLAP cubes [GCB+97] that provides a multi-dimensional view of the static data by aggregating attribute over different dimensions, mentioned beforehand. This method immensely helps to improve the efficiency of exploratory queries. It allows operations like drill-down, roll-up, and selection over OLAP cube to navigate and identify the data of users interests. Few more operators for OLAP cubes are introduced by Sarawagi et al. [SAM98, SS00] to guide a user to explore interesting regions of data, by highlighting the exceptions within it.

Further, to make OLAP smarter and more efficient for analyzing large dynamic data, different approaches that allow user interaction and feedback in OLAP operations are proposed in literature. Dash et al. [DRM+08] propose a dynamically faceted search system that automatically identifies a smaller subset of facets and values from large query results which are considered to be interesting to a user. Drosou and Pitoura [DP13] present a system, called YmalDB, that recommends additional dimensions to navigate the data, which are highly related to the original query given by users. The system first identifies interesting attribute-value pairs based on a frequency-based scoring model from the results of the initially-given user query. Then, these interesting pairs are used for attribute expansion and recommendation. Dimitriadou et al. [DPD14] propose the Automatic Interactive Data Exploration (AIDE) framework that learns interesting regions of the data from user feedback by using a decision tree. The authors also discuss the optimal way to generate sample data by capturing all relevant dimensions that are used to collect feedback from users. The work by Li et al. [LHY+08] also addresses the problem of applying OLAP over sample data. They present a framework, called Sampling Cube, that handle the error in result generation by intra- or inter-cuboid query expansion. Joglekar et al. [JGP16] propose an interaction operator to extend the scope of drill-down operations. It allows online user interaction and enables browsing the top-k most interesting explorative facts about the data, based on the dimensions preferred by the data analyst. Few other works focus on efficiency in exploration, by adopting distributed framework [KJTN14, AIP+12]. All these works, mentioned above, either collect hints from users to have a prior knowledge about a user's interest before initiating the exploration or collect statistics over the measuring attributes used for aggregating data over different dimensions, to identify interesting regions in data for the data exploration. In this thesis, we propose a different approach to identify the interesting facets, independently of the measuring attributes present in a table. Hence, our approach can be used orthogonally as an enabling step to approaches discussed earlier by providing recommendations of meaningful dimensions.

Wu et al. [WKQ+08] present an overview of ten extensively used mining algorithms, covering the area of classification, clustering, statistical learning, as-

sociation rule mining, and link mining. These algorithms use objective measures to discover hidden characteristics or patterns in the underlying data. Objective measures can efficiently identify the structural patterns of data, but fail to understand the interestingness of it. To deal with this problem, Silberschatz and Tuzhilin [ST95] coined the term 'subjective interestingness measures' which is expressed by two concepts, un-expectedness and actionability, and propose an approach to measure the interestingness based on the belief system of users. A mathematical framework, similar to the Bayesian inference model, is introduced by Bie [Bie11, Bie13] to formalize interestingness of a pattern, mined from the data. However, in this framework, user utility is considered an important parameter. There exist another set of works that train a classification model from training samples, collected from users feedbacks or query logs, in order to capture the user's interests in data exploration [DPD14, BLWJ15].

Avoiding direct user intervention, web data can be used to capture the trends or general interests of users. Mining web tables is a common practice for extracting such information. Knowledge bases like YAGO and DBpedia are created based on Wikipedia. The general purpose knowledge base ProBase [WLWZ12] is developed from web corpus based on a probabilistic approach that associates plausibility and typicality scores to every fact and relation. These knowledge bases facilitate a machine to understand human communication and concepts from the real world. Wang et al. [WWWZ12] present a prototype to associate the most suitable concept with table entities and their attributes by linking the entity with Probase. Cafarella et al. [CHW$^+$08] present a system, called WebTables, collecting 14.1 billion tables from web and enabling an effective search over the collected tables. Additionally, the authors discuss a model, called attribute correlation statistics database, which is used to provide facilities like schema auto-completion and attribute synonyms to explore the data. Yakout et al. [YGCC12] present a framework for augmenting web tables automatically based on topic-sensitive PageRank over the schema-matching graph on web tables, to reach higher coverage and discover important attributes of an entity type. Considering Wikipedia tables as a rich source of information, different question-answering frameworks are proposed in literature [SC14, BND13, CFWB17], for handling factoid questions. Bhagavatula et al. [BND13] propose a prototype that joins relevant Wikipedia tables in order to explore the correlation between attributes for an entity. As web tables are presented in diverse schema representations, extracting semantic interpretation of web table is a challenging task, addressed in [LSC10, RLB15, Zha17], and very important for enabling applications on top of it. The works on effective extraction of tables from web are orthogonal to the work present in this thesis, but can be adapted.

Fatima et al. [FCS17] introduce an approach, called DBpedia Trivia Miner, that can learn an interestingness model for a given domain. The approach combines features like tf-idf, mean, entropy of categories, and the automatically learned features using a neural network approach, to capture interesting facts about an entity. They characterize the interestingness of data by unusualness,

uniqueness and unexpectedness of the facts. The proposed model is trained over a manually annotated subset of DBpedia data. In this thesis, we also train a model that classifies interesting categorical attributes, where we harness Wikipedia tables to incorporate humans' perception of interestingness into our training data. In prior work [RPM16], we develop a scoring model for finding interesting rankings based on the statistics, computed over attributes that are used to create the ranking. This ad-hoc model involves a component to classify the interesting categorical attribute based on entropy gain measures, which is subsumed by our proposed model present in Chapter 6. As our work focuses only on the categorical attributes, we further discuss the literature on statistical measures applicable for categorical attributes.

3.3.1 Statistical Measures for Assessing Categorical Attributes

The survey by Geng and Hamilton [GH06] discuss different statistical measures that are used as metrics for describing 'interestingness' of a pattern discovered from data by classification, association rule mining, and summarization. The authors classified all state-of-the-art measures in three categories—objective, subjective, and semantic-based. They also provide an overview of the strategies of selecting measures for different applications. Both subjective and semantic-based measures require user involvement for defining the interestingness of a categorical attribute. Here, we discuss only the objective measures for categorical attributes as we aim to build a classification model that is capable of identifying interesting categorical attribute for an entity type, without any human involvement.

Dang and Croft [DC12] propose a new measure, called diversity by proportionality, that find diversity in the search results by comparing them to a predefined range of topics, associated with the query and the relevance of the results. In this thesis, we have adopted the similar concept of diversity, where the diversity of a categorical attribute is measured by comparing it with a predefined ideal distribution of an attribute. Different context-specific diversity measures are proposed to find the diversity of the search results, by combining relevance assessment of the results and its topic coverage [RBS10, AGHI09]. Schedl and Hauger [SH15] present a diversity measure, based on users profile of listening music, for creating a music recommender systems. Henzgen and Hüllermeier [HH14] present an analogy of support and interest measure from itemset mining to the context of mining subrankings. Hilderman and Hamilton [HH01] review thirteen diversity measures from different field of studies, for measuring the usefulness of a ranking summarization, generated from a database. They discuss five mathematical principles that can derive the usefulness of these interesting measure. Based on these principles, a comparative study among these diversity measures are presented in the paper. The study shows that only four measures; $I_{variance}$, $I_{shannon}$, $I_{simpson}$, and $I_{mcIntosh}$ fulfil all five principles. In this thesis, we also use shannon and simpson index for capturing interestingness

of a categorical attribute. Two other measures, unalikeability and peculiarity, discussed by Kader and Perry [KP07], are defined to capture variability of categorical data, used in this thesis.

3.3.2 Handling Imbalance Training Data using SVM

In this thesis, to create the classification model for identifying interesting categorical attributes, we employ an SVM over an automatically generated training data, as mentioned earlier in Section 1.2. Here, we observed that the generated training data is imbalance, i.e., the number of samples in one class are outnumbered by the other class. Many real-world classification applications in the area of information retrieval, genetics, etc. deal with highly imbalance training data. In this section, we briefly discuss existing methods to tackle this problem.

Schölkopf et al. [SPS$^+$01] discuss a new approach of SVM, called *one-class SVM* that can train a classification model from training data that either do not have any class information or belong to only one specific class. The idea behind one-class SVM is to find the decision function that returns '+1' for a small region capturing all points in the training set. They simply consider the origin as as the only candidate for the second class and create the boundary that maximizes the distance between origin and other data points. Manevitz and Yousef [MY01] propose a document classifier by using one-class SVM and present a comparative study of the performances among the proposed document classifier and other outlier classifiers such as Rocchio's algorithm, nearest neighbor, naïve Bayes, and compression neural network over *Reuters data set*. The study shows that the one-class SVM outperforms all other models except the neural network. The performance of computationally intensive compression neural network is comparable with one-class SVM in some scenarios. In this thesis, we also use one-class SVM, separately over positive and negative training sample, to deal with the imbalance training data.

Tax and Duin [AKJ04, TD01] present a support vector data description method to separate outliers from the data. It obtains a spherical bound over target data in a way that minimizes the outliers to be detected as target set. Allowing outliers at training time, this approach can create a more flexible description of training data. Wu and Chang [WC03b] present existing methods such as boundary movement, biased penalties [VCC99] to handle the problem of imbalance training data. Additionally, the authors propose a new approach, called class-boundary alignment, which tunes kernel functions in a way such that it maximizes the influence on the class boundary for the minority support vectors. Akbani et al. [AKJ04] propose another approach to deal with imbalanced training data, by combining biased penalties with oversampling of minority class while rejecting the undersampling of the majority instances to restrict the inherent loss due to undersampling. Tuning the cost of training samples from each class is a common strategy to deal with imbalance data, presented in [MBJ99, WLMJ14]. Wu and Chang [WC03a] discuss that Gaussian RBF kernel is a good choice for transforming data to feature space while using

class-boundary alignment approach to learn a model from imbalanced training data. In this thesis, we adopt the concept of undersampling method to work with the imbalance training data.

Chapter 4

Similarity Search over Rankings

4.1 Introduction

This chapter is based our own publications at WebDB 2014 [PM14] and SSDBM 2016 [PM16b]. In this chapter, we address the problem of performing efficient similarity search over top-k rankings; for a user-provided query ranking and similarity threshold. Finding similar rankings, in the sense that they report on a similar set of entities in roughly the same order, carries valuable analytical insights. The scope of similarity search over ranked lists is not only restricted to the applications in the area of information systems, but it is also common in other fields like behavioral studies in social sciences or studies on scientific data in experimental science, as mentioned in Chapter 1.

There exist several prominent distance functions, such as Kendall's Tau, Spearman's footrule distance, NDGC [JK02], the rank distance [Car09], and ERR [CMZG09], mentioned in Chapter 2 and Chapter 3. Considering simplicity, generalization, richness, and basic properties of these measures, the two predominant distance measures in literature are Kendall's Tau distance and Spearman's footrule distance, discussed in [KV10]. Both are originally defined over a pair of rankings that capture the same (full) domain of elements. For incomplete rankings of size k, Fagin et al. [FKS03] propose a generalization of Kendall's Tau and Spearman's footrule distances to handle the incompleteness of ranking domain within distance measures. However, Fagin et al. [FKS03] show that the generalized Kendall's Tau distance violates the triangle inequality property, which eliminates the possibility of using metric-space index structures, like the M-tree [CPZ97] in this work. There exist few spatial indexing methods like the R-Tree [Gut84] and the K-D Tree [Ben90] that can produce exact results for the NN-search, but are not efficient for high-dimensional data. Weber et al. [WSB98] show that the efficiency of NN-search using such structures be-

comes worse than a brute-force linear scan over all data when the dimensions become higher than ten. Therefore, the similarity search over top-k lists with the generalized Kendall's Tau distance requires further investigations on index structures.

In this chapter, we discuss the r-near-neighbor (r-NN) problem on top-k rankings using generalized Kendall's Tau distance. Harnessing the observation that at least one element should be contained in two given rankings to have a reasonable minimum similarity between them, one classical solution to r-NN search is employing an inverted index over each element of the rankings. Such an index is very efficient in answering set-containment queries [HM03]. As more overlapping elements between rankings signifies more similarity between them, we can use multiple elements together from rankings to create the inverted index.

Adapting to the characteristics of the Kendall's Tau distance, we present three different inverted index structures, using pairs or triplets of elements from rankings as the indexing granularity. When analyzing the performance of the query processing on a pairwise or triple index on real-world datasets, we observe that it is often *sufficient to query the index with a small subset of the query's items*, hence, leading to an approximate but very efficient query processing. To understand the reason behind this characteristics and theoretically derive the expected performance and the accuracy of the query processing on pairwise or triple index, we relate the concept to locality sensitive hashing (LSH). LSH is an efficient method for near-neighbor search (NN-search) [AI08] over high-dimensional data, discussed in Section 2.1.3. There exist LSH families for different metric distances, such as l_1, Euclidean, or Hamming distance [DIIM04, IM98]. Although the generalized Kendall's Tau is *not a metric distance function*, we observe that the generalized Kendall's Tau distance can be related to the Hamming distance and the Jaccard distance. Exploiting this connection, we propose here two LSH families for the generalized Kendall's Tau distance.

Considering that the data are uniformly distributed, LSH methods commonly select random hash functions from an LSH family to map the query data into the hash tables to find the similar objects. However, real-world data is prone to having a non-uniform distribution and it is difficult to capture characteristics of the query data without any prior knowledge or restrictive assumptions. Hence, we propose query-driven LSH methods that use a query-specific selection of hash functions to map the query into the hash table, to render LSH more efficiently for similarity search—without prior knowledge of the query data distribution. We also establish a one-to-one mapping of the pairwise and the triple index with the indices used in the proposed LSH methods. We compare the proposed indices to plain inverted indices, a full linear scan, and the state-of-the-art approach [WLF12] for querying for similar sets.

4.1.1 Problem Statement and Setup

Consider a set of top-k rankings $\mathcal{T}$, where each $\tau_i \in \mathcal{T}$ has a domain D_{τ_i} and $|D_{\tau_i}| = k$. The global domain of items is then $D = \bigcup_{\tau_i \in \mathcal{T}} D_{\tau_i}$. We investigate the impact of various choices of k on the query performance in the experiments. Figure 4.1 shows three example rankings.

$$\tau_1 = [2, 5, 4, 3, 1]$$
$$\tau_2 = [1, 4, 7, 5, 2]$$
$$\tau_3 = [0, 8, 7, 5, 6]$$

Table 4.1: Example rankings

Rankings are represented as arrays or lists of items; where the left-most position in the examples denotes the top-ranked item. The rank of an item i in a ranking τ is given as $\tau(i)$.

At query time, a user provides a query ranking q, where $|D_q| = k$ and $D_q \subseteq D$, a distance threshold θ_d, and a distance function d. Our objective is to find all rankings that belong to $\mathcal{T}$ and have a distance less than or equals to θ_d, i.e.,

$$\{\tau_i | \tau_i \in \mathcal{T} \wedge d(\tau_i, q) \leq \theta_d\}.$$

Using the item from rankings, we can build an inverted index to look up those rankings that have at least one item overlapping with the query's items, during query time. For the rankings found this way, the distance to the query is compared to find the final result. Considering the example in Figure 4.1, for a query ranking $q = [8, 7, 0, 6]$, ranking τ_1 does not overlap at all with the query's items, while τ_2 and τ_3 do overlap. Therefore, the Filter and Validate algorithmpresented in Chapter 2 can find τ_2 and τ_3 from the inverted index. According to the algorithm, the index is accessed for each of the query items to find the candidates. Then, the distance between the query and the candidates are calculated to retrieve the true results. Note that we assume the distance threshold θ_d to be strictly smaller than the maximum possible distance, i.e., the normalized maximum distance threshold $\theta < 1$. Thus, the inverted index can find all result rankings as all results need to have at least one overlapping item with the query.

In this work, we use the generalized Kendall's Tau distance as the distance function to deal with incomplete rankings. Moreover, we consider the penalty $p = 0$ for case 4 in Definition3 of the generalized Kendall's Tau distance, mainly because of two reasons—(1) it is difficult to predict the order of the missing elements if they appear in the top-k list and (2) their absence from one of the top-k list decreases their significance. We denote this distance function as K^0. Kendall's Tau is defined as the pairwise disagreement, which suggests building an inverted index using pairs of items as keys. Additionally, two ordered pairs can be combined into a triplet that also carries information about pairwise

disagreement. In this chapter, we present how exactly these indices are used in the r-NN search by relating them to proposed query-driven LSH schemes. We also show that they are in fact locality sensitive. Based on the proposed query-driven LSH schemes, we derive theoretical bounds on the number of index accesses that is required to reach a specific recall in the r-NN search.

4.1.2 Contributions and Outline

In this work, we make the following contributions:

- We first present ad-hoc similarity search over sets of top-k lists (aka. rankings), considering inverted indices over single items, pairs, and triplets. For every index structures, we derive how to find the minimum overlapping bound for a given generalized Kendall's Tau distance threshold based on prefix-filtering approach, which is used as the pruning rule to improve the efficiency of ad-hoc similarity search by eliminating false-positive results.
- We propose two different hash families for Kendall's Tau distance that facilitate query-driven Locality Sensitive Hashing (LSH) for Near-Neighbor search, resembling querying on pair- and triplet-based inverted index.
- We derive the theoretical bounds on the expected recall for the proposed query-driven LSH methods. We also describe how to automatically tune the number of hash table access in the proposed LSH schemes for a predefined recall requirement in NN-search.
- We compare the performance of the proposed LSH schemes to traditional inverted indices and to the SimJoin method—a competitor based on the popular prefix-filtering framework.

The rest of this chapter is organized as follows. Section 4.2 discusses the derivation of a distance bound for the plain inverted index. Section 4.3 shows the consequences of interpreting rankings as sets of pairs and triplets, which motivates to proposing LSH schemes for Kendall's Tau, discussed in Section 4.4. Section 4.5 presents the query-driven LSH method for the proposed LSH schemes and develops automated parameter tuning to increase the efficiency of similarity search. Section 4.6 discusses the performance of proposed LSH schemes compare to baseline approaches and presents the key findings. Finally, Section 4.7 summarizes the work of this chapter.

4.2 Inverted Index with Distance Bounds

As discussed earlier, it is possible to find similar rankings by retrieving all rankings from the inverted index that *overlap at least in one item* with the query items. We use such a basic inverted index on rankings, illustrated in Table 4.2 for Example rankings 4.1, as a baseline in the experimental evaluation.

$7 \rightarrow \langle\tau_2\rangle, \langle\tau_3\rangle$
$5 \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle, \langle\tau_3\rangle$
$4 \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$
...

Table 4.2: Basic Inverted Index

$(4,5) \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$
$(5,7) \rightarrow \langle\tau_2\rangle, \langle\tau_3\rangle$
$(3,4) \rightarrow \langle\tau_1\rangle$
...

Table 4.3: Sorted Pairwise Index

Applying **filter and validate** approach (Algorithm 1 in Section 2.1.2) on the basic inverted index, shown in Table 4.2, for a user-provided query ranking q and a distance threshold θ_d, we retrieve the candidates and the final results as follows:

- The inverted index is looked up for each element in D_q and a candidate set $\mathcal{C}$ of rankings is generated by collecting all distinct rankings seen in the accessed posting lists.
- For all such candidate rankings $\tau \in \mathcal{C}$, the distance function $K^{(0)}(\tau, q)$ is calculated and if $K^{(0)}(\tau, q) \leq \theta_d$ then τ is added to the result set $\mathcal{R}$.

Potentially, many of the candidate rankings in $\mathcal{C}$ are so-called *false positives*, i.e., rankings that are found while accessing the posting lists but do not belong to $\mathcal{R}$. Each such false positive causes an unnecessary distance function computation. Intuitively, $\tau \in \mathcal{R}$ should be found in at least a certain number of posting lists for becoming a potential candidate, depending on the distance threshold θ_d. In this section, we establish a criterion that allows removing some of the false positives by computing the minimal number of posting lists that need to be accessed, for the elements in D_q. This idea is similar to the *prefix-filtering* method [CGK06].

Lemma 4.2.1 determines the prefix-filtering parameter, i.e., the minimum overlap required between two rankings, so that the Kendall's Tau distance between them must lie within the threshold θ_d. This result is then used to propose Proposition 4.2.2 that derives the bound on the number of elements in D_q that are required to be looked up in the inverted index to retrieve all true results.

Lemma 4.2.1 *The* **minimum number of overlapping elements** *between two rankings τ and q to have at most a Kendall's Tau distance θ_d is given by $\mu = (k - \sqrt{\theta_d})$, where $|D_q| = k$.*

Proof: Let μ be the minimum number of overlapping elements between two rankings q and τ. Then, the *minimum* possible distance $K^{(0)}(\tau, q)$ with μ overlapping elements can be found by considering the following conditions:

(i) All overlapping elements are in the same order in both q and τ, which yields $\bar{K}^{(0)}_{i,j}(\tau, q) = 0$ for all $i, j \in \{D_\tau \cap D_q\}$, according to Case 1 in Definition 3 of the generalized Kendall's Tau distance.

(ii) All non-overlapping elements i of the ranking appear at the *bottom* of both lists, which yields $\bar{K}^{(0)}_{i,j}(\tau, q) = 0$ for all $i \notin \{D_\tau \cap D_q\}$ and $j \in \{D_\tau \cap D_q\}$, according to Case 2 in Definition 3 of the generalized Kendall's Tau distance.

Considering the above conditions, all the pairs (i, j), where $i \in D_\tau, j \in D_q$ causes $\bar{K}^{(0)}_{i,j}(\tau, q) = 1$, according to Case 3 in Definition 3 of the generalized Kendall's Tau distance. As each ranking has $(k - \mu)$ non-overlapping elements, the total number of such pairs is $(k - \mu)^2$.

In this thesis, we consider the penalty $p = 0$ for case 4 in Definition 3 of the generalized Kendall's Tau as mentioned, earlier in Section 4.1.1. Therefore, $\bar{K}^{(0)}_{i,j}(\tau, q) = 0$ for all pair $i, j \notin \{D_\tau \cup D_q\}$.

Hence, minimum $K^{(0)}(\tau, q) = (k - \mu)^2$. Now, bounding the minimum distance between two rankings to the threshold distance θ, we can bound μ, i.e., solving the equation $(k - \mu)^2 = \theta_d$, we get $\boldsymbol{\mu = \lceil k - \sqrt{\theta_d} \rceil}$. Clearly, all rankings with overlap $n < \mu$ will have $K^{(0)}(\tau, q) > \theta_d$, and therefore, can be immediately ignored. k^2 is the maximum distance possible between two top-k rankings, which we use to normalize the distance measure within [0,1], for example, normalized distance threshold $\theta = \theta_d / k^2$. Thus, μ can be also expressed by $\lceil k(1 - \sqrt{\theta}) \rceil$.

From the previous lemma, we can state that *all result rankings must appear in at least μ posting lists*, which leads to the following proposition.

Proposition 4.2.2 *For retrieving all rankings in $\mathcal{R}$, it is sufficient to look up the posting lists for only $k - \mu + 1$ elements from the query's domain D_q.*

Using Proposition 4.2.2 on the basic inverted index, we can perform a more efficient r-NN search under the generalized Kendall's Tau distance than the simple filter and validate approach. However, the concept behind the definition of Kendall's Tau distance opens the discussion of having slightly different index structures than the basic one, which we will introduce in next section.

4.3 Pairwise and Triple Index

Kendall's Tau is defined by discordant pairs between two rankings, which immediately suggests treating a ranking as a set of pairs. Further, we observe that triplets of items can also provide information about the order among item pairs. Therefore, we consider creating inverted indices using pairs and triplets as keys, respectively, which is feasible as top-k rankings are usually short compared to the potentially large global domain. In this section, we present three different ways to represent the rankings and propose three index structures based on those representations. Following the discussion from the previous section, we will also derive the pruning rules that allow us to avoid accessing the index for all pairs or triplets from the query's items during query processing.

4.3.1 Rankings as Sets of Pairs

Considering a ranking as a set of pairs allows us to directly compare overlapping pairs between two rankings to determine the Kendall's Tau distance. Representing a ranking τ as a set of sorted pairs τ_s^p, we define the sorted pairwise index as follows.

Definition 4 *The* **sorted pairwise index** *maps a pair $(i, j) \in \tau_s^p$ to a posting list that holds all rankings (ids) that contain both elements i and j, where τ_s^p represents all pairs of elements that occur in ranking τ, defined as:*

$$\tau_s^p = \{(i, j) | (i, j) \subseteq D_\tau \times D_\tau \wedge i < j\}.$$

The index structure is called sorted pairwise index because, according to the definition of τ_s^p, each key (i, j) follows $i < j$, without reflecting the order of appearance of i and j in τ. For instance, $\tau_{1s}^p = \{(2, 5), (2, 4), (2, 3), (4, 5), (3, 5), \ldots\}$ for the ranking τ_1 from Figure 4.1. Due to the ordering, redundant indexing is avoided. For clarification, Table 4.3 represents part of the sorted pairwise index for the example rankings given in Section 4.1.

In the τ_s^p representation, in order to compute the Kendall's Tau distance, overlapping pairs between the rankings need to be further verified to check the actual order of items in both rankings. Hence, we propose another pairwise representation of rankings as a set of unsorted-pairs τ_u^p, such that the order between items appears in the ranking can be preserved.

Definition 5 *The* **unsorted pairwise index** *maps each pair $(i, j) \in \tau_u^p$ to a posting list that holds all rankings (ids) that contain both elements i and j in this order, where τ_u^p represents all pairs of elements that occur in ranking τ, preserving their ranking order, defined as:*

$$\tau_u^p = \{(i, j) | (i, j) \subseteq D_\tau \times D_\tau \wedge \tau(i) < \tau(j)\}.$$

The index structure is called unsorted pairwise index because, according to the definition of τ_u^p, the keys in this index are maintaining the order in which the elements are appearing in the ranking. For example, $\tau_{1u}^p = \{(2, 5), (2, 4), (2, 3), (5, 4), \ldots\}$ for the ranking τ_1 from Figure 4.1. Hence, a posting list for key (i, j) in the unsorted pairwise index holds all rankings where i appears before j. Table 4.4 represents part of the unsorted pairwise index for the example given in Section 4.1.1. The space and time complexity for building both pairwise indices is $O(|\mathcal{T}|\binom{k}{2})$.

The simple filter and validate approach, introduced earlier for the single-item inverted index, can also be applied to process queries on these pairwise index structures. Unlike using each element from query, here, we look up the sorted or unsorted pairwise index for each pair $(i, j) \in q_s^p$ or q_u^p, respectively, to build the candidate set $\mathcal{C}$ in the filter procedure, presented in Algorithm 1. Then the retrieved candidates $\mathcal{C}$ are validated to find the result set $\mathcal{R}$.

$(5,4) \rightarrow \langle\tau_1\rangle$
$(7,5) \rightarrow \langle\tau_2\rangle, \langle\tau_3\rangle$
$(4,5) \rightarrow \langle\tau_2\rangle$
...

Table 4.4: Unsorted Pairwise Index

$(2,4,5) \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$
$(2,5,7) \rightarrow \langle\tau_2\rangle$
$(4,5,7) \rightarrow \langle\tau_2\rangle$
...

Table 4.5: Triple Index

We can generate $\binom{k}{2}$ possible pairs of items from query ranking q of size k, i.e., $|q_s^p| = |q_u^p| = \binom{k}{2}$. However, instead of probing the pairwise index for all $\binom{k}{2}$ pairs generated from q, we can use the prefix-filtering approach, discussed earlier. Now, we derive the minimum number of pairs from query, for which we need to look up both pairwise index structures to find all results, by adopting the Lemma 4.2.1 that state that a ranking needs minimum μ overlapping elements with the query to be qualified as a candidate. For pairwise indices, such μ elements generate $\binom{\mu}{2}$ pairs from a ranking. Therefore, all final results should contain at least $\binom{\mu}{2}$ pairs from q_s^p or q_u^p, respectively, for threshold θ_d. Hence, we obtain the following proposition.

Proposition 4.3.1 *For retrieving all rankings in $\mathcal{R}$, it is sufficient to look up the sorted or unsorted pairwise index for only $\binom{k}{2} - \binom{\mu}{2} + 1$ pairs of elements from q_s^p or q_u^p, for the given query q and similarity threshold θ_d.*

4.3.2 Rankings as Sets of Triplets

Based on the representation of a ranking as a set of triplets, denoted as τ^t for ranking τ, we propose another index structure, called triple index.

Definition 6 *The* **triple index** *maps a triplet $(a,b,c) \in \tau^t$ to a posting list that holds all rankings (ids) that contain all three elements a, b, and c, where τ^t represents all triplets of elements that appear in ranking τ, defined as:*

$$\tau^t = \{(a,b,c) | (a,b,c) \subseteq D_\tau \times D_\tau \times D_\tau \wedge a < b < c\}.$$

In this index, the elements in a key maintain the alphabetic order, and thus, avoid redundant indexing. For instance, $\tau_1^t = \{(2,4,5),(2,3,4),(3,4,5), \ldots\}$. Table 4.5 represents a part of the index created from the example given in Section 4.1. It is clear that the more elements are concatenated to create keys for an inverted index, the smaller the average size of a posting list becomes. Consequently, rankings mapped into the same posting list have a higher probability to be in fact similar. Therefore, a significant number of false positive candidates can be avoided by using larger keys. *On the other hand, the number of generated keys grows drastically with keys getting large.* For the triplets index, a total of $\binom{k}{3}$ keys are possible to create from a top-k ranking ranking τ, i.e., $|\tau^t| = \binom{k}{3}$. Hence, the space and building time complexity of triple index is $O(|\mathcal{T}|\ \binom{k}{3})$.

Again, the simple filter and validate paradigm can be applied for query processing in the triple index. Similarly to above, we can derive minimum

θ	0.1		0.2		0.3	
Access method	PF	M	PF	M	PF	M
Sorted Pairwise Index	25	3	31	4	36	6
Unsorted Pairwise Index	25	3	31	5	36	7
Triple Index	86	3	101	6	111	11

Table 4.6: Gap between prefix-filtering bound and manually tuned minimum required accesses

number of triplets a the query ranking that need to be used for retrieving the results $\mathcal{R}$. Adapting Lemma 4.2.1, all final results must appear in at least $\binom{\mu}{3}$ triplets from q^t, for a specific threshold θ_d. Thus, we obtain the following proposition.

Proposition 4.3.2 *For retrieving all rankings in $\mathcal{R}$, it is sufficient to look up the posting lists for only $\binom{k}{3} - \binom{\mu}{3} + 1$ pair elements from q^t for query q and specific θ_d.*

In practice, we can retrieve all results by accessing fewer pairs or triplets than the established bound, respectively, based on Proposition 4.3.1 or Proposition 4.3.2. For example, Table 4.6 shows a study on a real-world dataset of top-10 rankings created from Yago. The column **PF** represents the number of access derived by Proposition 4.3.1 and Proposition 4.3.2 for pairwise indices and triple index. The column **M** represents the minimum number of accesses required to retrieve all the results, tuned manually. From the Table 4.6, we can observe the clear distinction between the prefix-filtering bound and manually tuned number of accesses. For a small distance threshold ($\theta \leq 0.3$), in the prefix-filtering method, the triple index is accessed *at least ten times more* than the actual minimum access requirement. Similarly, the prefix-filtering method accesses pairwise indices *at least five times more* than the actual minimum access requirement to find all the results. According to the concept of prefix-filtering method, if we access less number of index entries than the access bound proposed by the approach, we will no longer guarantee the retrieval of 100% result, i.e., the 100% recall. However, compromising the 100% recall requirement, we can access less number of index entries and still hope that we will retrieve very many results, not a scientific way to realize the problem. Hence, we want to find out how the violation of prefix-filtering bound affects the recall of the similarity search, which will help us to probe the fewer index entries while satisfying a high recall. To understand this characteristic and derive a more strict bound on the number of access for proposed index structures, we relate these indices with LSH, discussed in the next Section 4.5 and verified by the experimental study in Section 6.4.

4.4 LSH Schemes for Kendall's Tau

Candidate rankings fetched from the above indices are eventually evaluated to reveal the actual results similar to the query. Particularly, the pairwise and the triplet-based approach access the index far more times than the actual accesses needed, depending on the value of k. On the other hand, larger keys render the lookups more "precise". In this section, we relax the problem of r-NN search by adopting the concept of LSH to find a strict bound on the number of required lookups that still allows achieving a high recall value close to 100%. Here we present how exactly such an approximate querying can be assessed by **relating the discussed indices to LSH schemes.** Specifically, we propose two hash families for Kendall's Tau distance and show that they are in fact locality sensitive. For each of these families, we consider two hashing schemes and theoretically evaluate their suitability to efficiently retrieve the query results.

4.4.1 Hash Family 1

We introduce the first hash family, denoted as $\mathcal{H}_1$ that contains projections respect to all distinct elements from the global domain $D = \cup_{\tau \in \mathcal{T}} D_\tau$. Each hash function divides the data space into two half spaces.

Definition 7 *$h_i \in \mathcal{H}_1$, with $i \in D$, is defined as*

$$h_i(\tau) = \begin{cases} 1, & i \in D_\tau \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Clearly, the hash functions from $\mathcal{H}_1$ project rankings to binary space based on the presence of individual elements, e.g., $h_2(\tau_1) = 1$, $h_2(\tau_3) = 0$, considering the example from Section 4.1.

For a distance threshold θ_d, if two objects lies within the distance θ_d, a LSH family ensures that those two objects have higher probability to collide into the same hash bucket than other objects that have distance strictly higher than θ_d, according to Definition 1. Consider, P_1 is the probability that similar objects collide into the same bucket and P_2 is the probability that two dissimilar objects collide into the same bucket, according to $\mathcal{H}_1$. To show that the hash family $\mathcal{H}_1$ is locality sensitive, we have to prove that $P_1 > P_2$.

Locality Sensitivity of Hash Family 1

Each $h_i \in \mathcal{H}_1$ maps $\tau, q \in \mathcal{T}$ to $\{0, 1\}$ according to the presence of i in τ and q. A collision into bucket '1' signifies the overlapping of one element between τ and q. Hence, we can directly relate this hash family with the overlapped-based distance measure like Jaccard distance . As this work considers Kendall's Tau distance, we first find the equivalent Jaccard distance threshold corresponding to the Kendall's Tau distance threshold. Then the approximated Jaccard distance

threshold is used as r to find the collision probability P_1 of hash family $\mathcal{H}_1$. For the notation, throughout this chapter, P_1, P_2, c, and r refer directly to Definition 1. In Section 4.2, we have already derived the required minimum overlap μ for two rankings to have a chance to satisfy the Kendall's Tau distance threshold θ_d. Therefore, the Jaccard distance with minimum μ overlaps, i.e., $1-\mu/(2k-\mu)$, becomes the approximated equivalent Jaccard distance threshold (r) corresponding to the Kendall's Tau threshold θ_d. Following the discussion, it is clear that the probability $Pr[h_i(q) = h_i(\tau)]$ is the same as the Jaccard similarity $\mu/(2k-\mu)$, if $i \in \{D_q \cup D_\tau\}$. As the $\mathcal{H}_1$ is defined over the global domain D, all hash functions h_i, where $i \notin D_q \cup D_\tau$, map q and τ to the same bucket (labeled as '0') as $i \notin D_q, D_\tau$. Let $|\overline{D_q \cup D_\tau}| = \lambda$, then, $|D| = 2k - \mu + \lambda$, and the collision probability is then $P_1 = (\mu + \lambda)/|D|$. More similar rankings will have more overlapping elements than μ, which increases the collision probability P_1, i.e., more similar rankings have higher chance to be placed in the same bucket.

On the other hand, as long as the approximation factor[1] c is strictly larger than 1, rankings that have distance larger than cr, have strictly less than μ overlapping elements. Hence, we can say $P_2 < P_1$. Thus, *the locality sensitive property holds for* $\mathcal{H}_1$.

Function Families for Hash Family 1

Based on the hash family $\mathcal{H}_1$, we propose two LSH function families $\mathcal{G}_1$ and $\mathcal{G}_2$ in a way such that the implementation of hash tables for these function families can directly use the proposed pairwise and triplets-based index.

Scheme 1 is based on function family $\mathcal{G}_1$, which is created by concatenating two hash functions, defined as follows:

$$\mathcal{G}_1 = \{(h_i, h_j) | (h_i, h_j) \in \mathcal{H}_1 \times \mathcal{H}_1 \text{ and } i < j\}$$

Hence, a function $g \in \mathcal{G}_1$, where $g = (h_i, h_j)$, projects a ranking τ to the space $\{0,1\}^2$. Using the example rankings from Table 4.1, let us consider a function $g_1 = (h_4, h_5) \in \mathcal{G}_1$, as $4 < 5$. Hence, $g_1(\tau_3) = (0,1)$, $g_1(\tau_1) = g_1(\tau_2) = (1,1)$. Here, we notice that the bucket labeled as $(1,1)$ for g_1 contains both τ_2 and τ_3 which is similar with the bucket labeled $(4,5)$ in the sorted pairwise index illustrated in Table 4.4. Clearly, the bucket label $(1,1)$ for a hash function $g = (h_i, h_j)$ is represented by the key (i,j) in the sorted pairwise index. Thus, the sorted pairwise index contains all hash table entries with bucket label $(1,1)$ from all the hash tables introduced by $\mathcal{G}_1$. Now, looking up the index for l pairs $(i,j) \in \tau_s^p$ implies applying l different functions $g \in \mathcal{G}_1$ on τ.

Further, we define Scheme 2 with function family $\mathcal{G}_2$ by combining three hash functions from $\mathcal{H}_1$, according to the following definition:

$$\mathcal{G}_2 = \{(h_a, h_b, h_c) | (h_a, h_b, h_c) \in \mathcal{H}_1^3 \text{ and } a < b < c\}.$$

[1] A discussion on the value of c is out of scope for this work as this work does *not* address the c-approximate nearest neighbor search problem.

	$h_{2,5}$	$h_{4,5}$	$h_{3,4}$	$h_{3,7}$	...
τ_1	1	0	0	1	...
τ_2	0	1	0	0	...

Table 4.7: Projections of rankings under $\mathcal{H}_2$

Here, $g \in \mathcal{G}_2$, where $g = (h_a, h_b, h_c)$, projects a ranking τ to $\{0,1\}^3$. Similar to the case for $\mathcal{G}_1$, looking up a triplet from a ranking τ means accessing the bucket with label $(1,1,1)$ in the hash table of $g = (h_a, h_b, h_c)$, where $a, b, c \in D_\tau$. Therefore, it is clear that the triple index contains only $(1,1,1)$ bucket labels from all hash tables that are introduced by $\mathcal{G}_2$. Thus, looking up the triple index for l triplets $(a,b,c) \in \tau^t$ implies applying l different functions $g \in \mathcal{G}_2$ on τ.

The query performance for different values of l is investigated in the experimental evaluation in Section 6.4, for both function families.

4.4.2 Hash Family 2

Instead of considering hash functions based on projections on individual elements, we now define a hash family $\mathcal{H}_2$ that contains all projections on $D_\mathcal{P}$, where $D_\mathcal{P} = \{(i,j) | (i,j) \in D \times D \text{ and } i < j\}$, i.e., all sorted ordered pairs over D. Hence, $\mathcal{H}_2 = \{h_{i,j} | (i,j) \in D_\mathcal{P}\}$.

Recall from Chapter 2 that the generalized Kendall's Tau is defined by the number of total discordant pairs in the domain $D_\tau \cup D_q$, for a ranking τ and query ranking q. To resemble this definition/behavior via hash functions, we define hash functions $h_{ij} \in \mathcal{H}_2$ that project τ on $\{0,1\}$ as follows:

Definition 8 *$h_{i,j} \in \mathcal{H}_2$, with $(i,j) \in D_\mathcal{P}$, is defined as*

$$h_{i,j}(\tau) = \begin{cases} 1, & i,j \in D_\tau \text{ and } \tau(i) < \tau(j) \\ 1, & i \in D_\tau \text{ and } j \notin D_\tau \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

Unlike $\mathcal{H}_1$, this hash family is reflecting the order of elements in a ranking with respect to the order of elements in the domain $D_\mathcal{P}$. For example $h_{1,4}(\tau_1) = 0$, $h_{1,4}(\tau_2) = 1$, considering the example from Section 4.1.

Locality Sensitivity of Hash Family 2

After projecting a ranking on $D_\mathcal{P}$, the ranking can be represented as a string of $\{0,1\}$, where '0' or '1' represents the bucket label on which the ranking is mapped by a specific hash function from $\mathcal{H}_2$. For clarification, such a representation for τ_1 and τ_2 under hash family $\mathcal{H}_2$ is shown in Table 4.7. Here, we can see that the example rankings τ_1 and τ_2 from Table 4.1 are mapped into different buckets for hash function $h_{2,5}$, according to the first rule of the definition of $\mathcal{H}_2$

(Definition 8). Relating this scenario to the generalized Kendall's Tau distance, $\bar{K}^{(0)}_{2,5}(\tau_1, \tau_2) = 1$, according to Case 1 in Definition 3 as the ranking order of the pair (2,5) is different in τ_1 and τ_2. Considering function $h_{3,4}$, both rankings τ_1 and τ_2 are mapped to the same bucket with label '0', according to the second rule of the definition of $\mathcal{H}_2$. This scenario is identical to Case 2 in Definition 3, which says $\bar{K}^{(0)}_{3,4}(\tau_1, \tau_2) = 0$. Considering a scenario of Case 3 in Definition 3, $\bar{K}^{(0)}_{3,7}(\tau_1, \tau_2) = 1$ and we find that $h_{3,7}$ also maps τ_1 and τ_2 into different buckets using second and third rule of the definition of $\mathcal{H}_2$, respectively. In summary, according to the definition of $\mathcal{H}_2$, two rankings do not collide into the same bucket for a hash function $h_{i,j} \in \mathcal{H}_2$ if the pair (i, j) is a discordant pair according the definition of generalized Kendall's Tau. Hence, the generalized Kendall's Tau distance $K^{(0)}(\tau_1, \tau_2)$ is the *hamming distance* between the binary representation of $\tau_1 and \tau_2$, introduced by $\mathcal{H}_2$. Consequently, the probability that a query q and ranking τ collide into the same bucket, i.e., $Pr[h(q) = h(\tau)]$, is the fraction of projections on $D_{\mathcal{P}}$ for which τ and q agree. Following the discussion above, two rankings will not be projected into the same bucket if the rankings are projected on those sorted pairs that are responsible for the distance. Hence, we obtain the collision probability $P_1 = 1 - {\theta_d}/{|D_{\mathcal{P}}|}$ considering r as θ_d. If the distance between rankings is smaller than θ_d then P_1 increases, i.e., if rankings are more similar then the probability to project those ranking into same bucket is larger. As long as $c > 1$, we have $P_1 > P_2$ and thus the property of locality sensitive hashing holds for $\mathcal{H}_2$.

Function Families for Hash Family 2

Based on $\mathcal{H}_2$, we define two function families $\mathcal{G}_3$ and $\mathcal{G}_4$ that use the previously introduced unsorted pairwise index. We propose Scheme 3 with function family $\mathcal{G}_3$, defined by selecting any hash function over $\mathcal{H}_2$, i.e.,

$$\mathcal{G}_3 = \{h_{i,j} | h_{i,j} \in \mathcal{H}_2\}.$$

For a $g \in \mathcal{G}_3$, i.e., $g = \{h_{i,j}\}$, the bucket labels '1' and '0' of g are represented respectively by the key element (i, j) and (j, i) in the unsorted pairwise index. According to this interpretation of hash bucket labels, the unsorted pairwise index holds hash table entries for all hash functions in $\mathcal{H}_2$. Hence, the ranking lists corresponding to a key (a, b) from the unsorted pairwise index is the same as the bucket label '1' for a function $g = h_{i,j}$ with $\{i, j\} = \{a, b\}$. Therefore, looking up the unsorted pairwise index for l pairs from τ^p_u implies applying l different functions $g \in \mathcal{G}_3$ on the query ranking.

We define Scheme 4 with function family $\mathcal{G}_4$ by combining two hash functions from $\mathcal{H}_2$, according to the following definition.

$$\begin{aligned}\mathcal{G}_4 = \{&(h_{i,j}, h_{x,y}) | (h_{i,j}, h_{x,y}) \in \mathcal{H}_2 \times \mathcal{H}_2, \\ &(i, j), (x, y) \in D_{\mathcal{P}} \text{ and } |\{i, j\} \cap \{x, y\}| < 2\}.\end{aligned}$$

Hence, function $g \in \mathcal{G}_4$, with $g = (h_{i,j}, h_{x,y})$, projects a ranking τ to $\{0, 1\}^2$. From the discussion of Scheme 3, we know how the hash table for $h_{i,j} \in \mathcal{H}_2$

is related to the unsorted pairwise index. Following that discussion, we can easily retrieve the entry of the hash table for $g \in \mathcal{G}_4$ by intersecting the posting lists from the unsorted pairwise index for key-pair used in g. For example, if $g = (h_{i,j}, h_{x,y})$ maps a ranking τ to $(0, 1)$, we can retrieve the rankings appearing in the same bucket with τ by intersecting the ranking lists retrieved for the keys (j, i) and (x, y) from the unsorted pairwise index—note that here we scan key (j, i) and not (i, j) as $h_{i,j}$ maps τ to 0. Applying l different $g \in \mathcal{G}_4$ functions on the query element relates to looking up l different key pairs in the unsorted pairwise index.

The impact of l on both function families are studied theoretically in the following section and evaluated in the experiments in Section 6.4.

4.5 Query-Driven LSH

It is clear from the discussion in the previous section that both hash families $\mathcal{H}_1$ and $\mathcal{H}_2$ contain a large number of hash functions. Both project a ranking in binary space according to the presence or absence of an element (in case of $\mathcal{H}_1$) or a pair of elements (in case of $\mathcal{H}_2$) in the ranking. Moreover, we consider top-k rankings where $k << |D|$. Hence, $\mu << |D|$ and the Kendall's Tau distance threshold $\theta_d << |D_{\mathcal{P}}|$. Consequently, the bucket that contains rankings projected on the *absence* of an element or a pair of elements (i.e., a bucket with label '0') holds very many entries than the bucket that contains rankings projected on the *presence* of an element or a pair of elements (i.e., a bucket with label '1'). This non-uniform distribution of rankings in hash buckets suggests to investigating how the selection of hash functions can optimize the number of retrieved candidate rankings.

A prominent intuition is that a larger number of shared elements between a ranking and a query increases the possibility of the ranking being similar "enough" to the query. Hence, in practice, we consider those hash functions that project the query only on the elements or pair of elements that are present in the query. Using this strategy, we can avoid the projection of the query into less informative and large hash buckets. Consequently, it helps to collecting fewer number of candidates, and thus, increases the efficiency of the LSH method by avoiding the validation of many false positives. As mentioned in the introduction of this chapter, real-world data are not always uniformly distributed over rankings. Therefore, selectively choosing hash functions from the hash families, based on the elements from queries can render the proposed LSH method more efficient than a basic LSH method. Likewise, to improve the efficiency of our proposed LSH methods, we propose a query-driven LSH method that choose hash functions from proposed hash families, biased by the elements appeared in the query, during query processing. Such adaptive nature of this strategy does not require analyzing the distribution of elements in queries beforehand. In the next section, we will discuss how exactly this biased choice of hash functions affect the collision probability of the proposed LSH schemes.

4.5.1 Refining the Collision Probability and Tuning l

According to query-driven LSH, for $\mathcal{H}_1$, all the hash functions which are selected to map a query into hash tables during query processing are projections on the elements from the query ranking. Similarly, for $\mathcal{H}_2$, all the hash functions which are selected to map a query into hash tables during query processing are projections on ordered pairs from the query ranking. Hence, the collision probability changes for both proposed hash families.

The collision probability P_1 for $\mathcal{H}_1$ becomes μ/k because any one of the shared elements is drawn from a total of k query elements. Based on this refined P_1 values, the tuning of parameter l can be determined by fixing error probability δ. According to the definition of $\mathcal{G}_1$ and $\mathcal{G}_2$, the parameter m is two and three respectively for Scheme 1 and Scheme 2. By substituting P_1 and m in Equation 2.1, we obtain the recall functions for Scheme 3 and Scheme 4 with error probability δ as follows.

$$1 - \delta \geq 1 - (1 - (\mu/k)^2)^l \qquad \text{Scheme 1} \qquad (4.3)$$

$$1 - \delta \geq 1 - \left(1 - (\mu/k)^3\right)^l \qquad \text{Scheme 2} \qquad (4.4)$$

For $\mathcal{H}_2$, all the pairs that are responsible for the θ_d distance between a ranking and a query are drawn from a total of $\binom{k}{2}$ ordered pairs from the query. It is important to notice here that the discordant pairs between two lists according to Case 3 in Kendall's Tau Definition 3 include only single element from each lists and such pairs are not contained in $\binom{k}{2}$ pairs created from the query. Case 3 is responsible for $(k-x)^2$ distance for x overlapping elements between two rankings. According to Lemma 4.2.1, a minimum of μ overlapping elements are required for the distance threshold θ_d. Therefore, the distance $(k-\mu)^2$ is not introduced by the pairs possible to form using only the elements from a query. Clearly, the pairs created only from the query's elements can be responsible for only $(\theta_d - (k-\mu)^2)$ distance. This distance is the lower bound of the number of discordant pairs drawn from the query. As the number of overlapping elements becomes more than μ, we can derive more than $(\theta_d - (k-\mu)^2)$ discordant pairs solely based on the query. Let $E(\mu)$ denotes the expected number of overlapping elements for a ranking that lies within θ_d distance from the query q. Then, for hash family $\mathcal{H}_2$, the collision probability P_1 becomes $1 - (\theta_d - (k - E(\mu))^2 / \binom{k}{2})$.

Based on this refined P_1 value for hash family $\mathcal{H}_2$, we find the parameter l by fixing error probability δ. According to the definition of $\mathcal{G}_3$ and $\mathcal{G}_4$, the parameter m is one and two, respectively for Scheme 3 and Scheme 4. By substituting P_1 and m in Equation 2.1, we obtain the recall functions for Scheme 3 and Scheme 4 as follows.

$$1 - \delta \geq 1 - \left(1 - \left(1 - \frac{\theta_d - (k - E(\mu))^2}{\binom{k}{2}}\right)\right)^l \qquad \text{Scheme 3} \qquad (4.5)$$

$$1-\delta \geq 1-\left(1-\left(1-\frac{\theta_d-(k-E(\mu))^2}{\binom{k}{2}}\right)^2\right)^l \qquad \text{Scheme 4} \qquad (4.6)$$

A comparison among the recall functions for all four schemes based on Equations 4.3–4.6 will not be appropriate as the distance threshold used in Scheme 1 and Scheme 2 is the Jaccard distance represented in terms of μ, where μ gives the lower bound of overlapping elements required to meet Kendall's tau distance threshold θ_d. Therefore, to compare all schemes, we modify the recall functions for Scheme 3 and Scheme 4 by expressing the collision probability P_1 for $\mathcal{H}_2$ in terms of μ. Hash family $\mathcal{H}_2$ projects rankings on ordered pair elements from $D_{\mathcal{P}}$, explained in Section 4.4.2. In query-driven LSH, considering at least μ overlapping elements exist between a ranking and a query, all the pairs that disagree between them must be drawn from $\binom{k-\mu}{2}$ pairs from the query. So, the probability that a pair of elements, appearing in both rankings, becomes $(1-\binom{k-\mu}{2}/\binom{k}{2})$. But these pairs have 50% chance to be appeared in the same order in both rankings. Hence, for $\mathcal{H}_2$, the collision probability P_1 can also be represented by $0.5(1-\binom{k-\mu}{2}/\binom{k}{2})$. Using this P_1, the recall functions for Scheme 3 and Scheme 4, presented by Equation 4.5 and 4.6, are reformulated and given by Equation 4.7 and 4.8, in terms of μ.

$$1-\delta \geq 1-\left(1-0.5\left(1-\frac{\binom{k-\mu}{2}}{\binom{k}{2}}\right)\right)^l \qquad \text{Scheme 3} \qquad (4.7)$$

$$1-\delta \geq 1-\left(1-0.25\left(1-\frac{\binom{k-\mu}{2}}{\binom{k}{2}}\right)^2\right)^l \qquad \text{Scheme 3} \qquad (4.8)$$

Figure 4.1 shows the comparison among the four LSH Schemes, given by Equations 4.3, 4.4, 4.7, and 4.8. It presents how recall increases as l increases, for threshold $\theta = 0.1$ and $k = 10$.

A predefined recall value $(1-\delta)$, determined by given error probability δ, for all proposed schemes can be reached by varying m and l. In our proposed schemes, as we have fixed the m-value, the recall can be assured by tuning the parameter l, i.e., the number of hash tables we need to access to retrieve all candidates for the similarity search. For instance, Figure 4.1 shows that Scheme 1 and Scheme 3 can reach 99.99% recall ($\delta = 0.0001$) with $l = 8$ for $\theta = 0.1$ (i.e., $\mu = 7$). This means that 8 hash tables (i.e., 8 entries from both pairwise indices) are enough to retrieve 99.99% of the true positive candidates. From Figure 4.1, we can see that this bound (l) is more than three times lower (to obtain 100% recall) than the bound established earlier in Proposition 4.3.1, discussed in Section 4.3.1, which would need to look up 25 entries. This improvement is consistent for the other schemes, too.

The actual tighter bound for l can be calculated by solving Equations 4.5 and 4.6, as these equations use the exact Kendall's tau distance threshold θ_d in

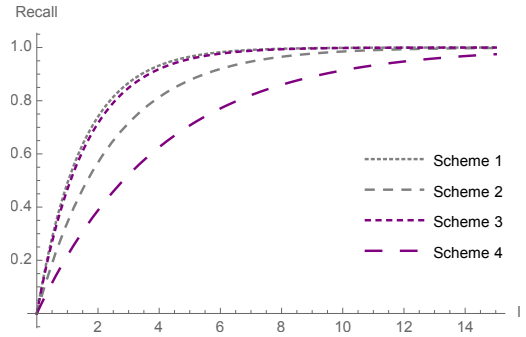


Figure 4.1: Comparison of recall functions among query-driven LSH schemes; $k = 10$.

	k=30				**k=40**			
θ	0.1	0.2	0.3	0.4	0.1	0.2	0.3	0.4
InvIn+Drop	10	14	17	19	13	18	22	26
Scheme 3	3	4	6	10	3	4	6	8
Scheme 4	4	7	14	33	4	7	14	22

Table 4.8: Lower bound of parameter l

calculation of collision probability rather using relaxed threshold using μ, discussed earlier. Depending on δ, the value of l can be higher than $\binom{k}{2}$. Hence, we consider accessing $min(\binom{k}{2}, l)$ hash tables in the query-driven LSH to avoid selecting the same hash function multiple times. Table 4.8 presents the number of hash tables (keys in pairwise and triple index) that need to be scanned compared to the access in inverted index using Proposition 4.2.2 (it is denoted as the baseline InvIn+Drop), for $\delta = 0.01$. We see that our proposed methods scan significantly a fewer entries compared to InvIn+Drop for lower θ, and subsequently, the prefix-filtering approach on the pairwise and triple index. The difference between the index access using the proposed LSH and the baseline InvIn+Drop becomes large as the ranking size k increases, and thus, making our schemes more efficient.

In practice, we observe that even 100% recall can be obtained with l tuned for $\delta = 0.01$ (i.e., recall 99.99%), as shown in Table 4.8. We further discuss this in Section 4.6.2 for top-10 and top-20 rankings. Figure 4.1 shows that the recall functions for Scheme 1 and Scheme 2 give an upper bound for the recall functions Scheme 3 and Scheme 4, respectively. For these characteristics, the parameter l that is determined for Scheme 3 and Scheme 4 can be used for Scheme 1 and Scheme 2, respectively, to achieve a predefined recall value. This is validated by our experimental results in Section 4.6.2. Hence, the tighter bounds of l determined from Scheme 3 and Scheme 4 are used to tune the parameter l in query-driven LSH schemes.

4.5.2 Effect of the Position of Query Elements

According to Definition 3 of the generalized Kendall's Tau distance, we notice that a non-overlapping element appearing at a top rank is responsible for more discordant pairs than if it would appear at a lower rank. Hence, the list with more non-overlapping elements in the top of a ranking list is more unlikely to satisfy the similarity threshold. Here, we discuss how this characteristic affects the candidate pruning during similarity search. We denote the total number of discordant pairs caused by any non-overlapping element that appears in the i^{th} position as $\bar{K}_i^{(0)}$. For any two top-k rankings and a given threshold θ_d, the upper and lower bound of the $\bar{K}_i^{(0)}$ can be calculated by following equations.

$$_{up}\bar{K}_i^{(0)} = min(k-i, \mu) \quad \text{where } \mu = k - \sqrt{\theta_d} \tag{4.9}$$

$$_{low}\bar{K}_i^{(0)} = \begin{cases} 0 & i \geq \mu \\ \mu - i + 1 & \text{otherwise} \end{cases} \tag{4.10}$$

$$_{avg}\bar{K}_i^{(0)} = 1/2(_{low}\bar{K}_i^{(0)} +_{up} \bar{K}_i^{(0)}) \tag{4.11}$$

According to Equations 4.9 and 4.10, Figure 4.2 shows how the rank of a non-overlapping element contributes to the distance, expressed in percentage of distance threshold θ. From Figure 4.2, we observe that missing elements from very top positions is responsible for more than 50% discordant pairs of the total Kendall's Tau distance for a small distance threshold (such as $\theta = 0.1$). Consequently, we can say that the pruning of false positive candidates can be achieved more effectively for the hash functions that project the rankings on the elements from top positions, for a small distance threshold.

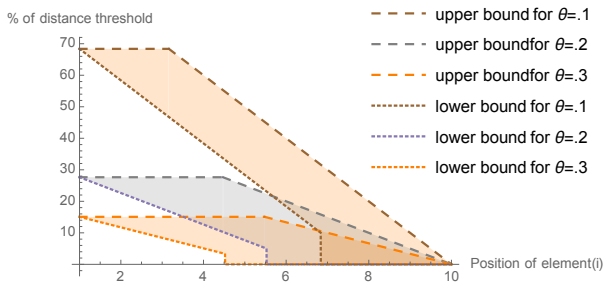


Figure 4.2: Number of discordant pairs due to different positions (rank) of elements in a top-10 ranking

Therefore, selecting hash functions that project on elements from the topmost positions have a higher probability to retrieve true positive candidates.

In Figure 4.2, we also see that the lower and the upper bound of contribution to the distance for the i^{th} non-overlapping element decreases as the threshold

increases. Hence, selecting hash functions by giving preference to top-ranked elements becomes comparatively less effective in candidate pruning for larger distance thresholds. Additionally, our proposed LSH schemes use pairs or triplets from the query instead of using distinct elements for projecting rankings, and thus, focusing on only top elements will have a higher chance to retrieve duplicate rankings as candidates. Here, we present Algorithm 3 to select l functions for query-driven LSH schemes in a way that ensures the selected hash functions projecting the query *not emphasizing only* the top elements of the query.

Algorithm 3: Selecting l functions for the query-driven LSH schemes

Data:
$q = \{e_1, e_2, \ldots, e_k\}; q(e_i) =$ position of e_i
weight of $e_i, W(e_i) =_{avg} \bar{K}_i^{(0)} / \sum_{e_j \in q} {}_{avg}\bar{K}_i^{(0)}$
$\mathcal{H} = \{h_1, h_2, \ldots\}; W(h_i) = \sum_{e_j \in E(h_i)} W(e_j)$
$\mathcal{G} = \{g_1, g_2, \ldots\}$; $W(g_i) = \sum_{h_j \in g_i} W(h_j)$
$E(h_x) =$ elements used in h_x for projecting q, $h_x \in \mathcal{H}$

1 **Procedure** `selectHash`$(q, W, \mathcal{H}, \mathcal{G})$
2 $S := \emptyset$
3 $S := argmax_{g_i}\{W(g_i) | h_j \in g_i \wedge E(h_j) \in q\}$
4 **while** $|S| \leq l$ **do**
5 $\mathcal{G} := \mathcal{G} - S$
6 $T := argmin_{g_i}\{\sum_{e_a, e_b \in \bigcup_{h_x \in g_i}} |q(e_a) - q(e_b)|$ where $h_j \in g_i \wedge E(h_j) \in q\}$
7 $S := S \cup argmax_{g_i}\{W(g_i) | g_i \in T\}$
8 **end while**
10 **return** S

If we replace Line 6 in Algorithm 3 with $T = \{g_i | h_j \in g_i \wedge E(h_j) \in q\}$, the algorithm will return l functions that *only* emphasize on top elements from the query. In Section 4.6.3, a comparison between random selections of hash functions and position-influenced selections is discussed with experimental results for real-world datasets.

4.6 Experimental Evaluation

We have implemented the index structures described above in Java 1.6 and run the experiments using an Intel Xeon CPU @ 2.67GHz machine, running Linux kernel 3.2.60, with 264GB main memory. The index structures are kept entirely in memory.

To evaluate the querying performance in terms of **query response time** (here, wallclock time), **number of retrieved candidates**, and **recall** (i.e., fraction of results found), we use two different datasets.

Yago Entity Rankings: This dataset contains 25,000 top-k rankings that

have been mined from the Yago knowledge base, as described in [IMS13]. Essentially, it contains entity rankings like the top-10 tallest buildings in the world or the countries by population.

NYT: This dataset contains search engine result rankings, created using 1 million keyword queries, randomly selected out from a published query log of a large US Internet provider. These queries are executed against the New York Times annotated corpus [San08] using a standard tf*idf scoring model with Dirichlet smoothing from the information retrieval literature.

Characteristics of the mentioned datasets are different, specifically, considering the frequency in which items appear across different rankings. The Yago dataset holds real-world entities where each entity occurs in a few rankings only. On the other hand, the NYT dataset comprises many popular documents that appear in many query-result rankings. Considering that the distribution of items in rankings follows Zipf's law, the estimated skewness parameter for NYT and Yago datasets are 0.87 and 0.53, respectively.

In addition to the baseline approach described in Section 4.2, we consider a competitor based on the work by Wang et al. [WLF12]. Based on the concept of prefix-filtering, the authors propose an adaptive filtering framework for improving the performance of executing joins between two sets. This framework considers a global order of all elements in sets to build a *Delta Inverted Index* that uses different prefix parameters to index the elements in sets. Then, they propose an algorithm for similarity search, called SimJoin Query, on top of their adaptive framework, coined as it Adapt-Join. We implemented their proposed index structure *Delta Inverted Index*, where the global order of the items in rankings is generated by the tf*idf measure of each elements appearing in rankings, and then apply *SimJoin* over the built index.

The runtime performance is measured in terms of average runtime for 1000 queries while varying the *normalized* distance threshold θ (given by $\theta_d = k^2 \times \theta$). Overall, a comparative study on the following approaches is presented:

- The filter and validate method on the single item-based inverted index denoted as **InvIn**.
- Prefix-filtering method on the single-item-based inverted index, where larger posting lists are dropped selectively, using the distance bound given in Proposition 4.2.2, denoted as **InvIn+Drop**.
- The presented LSH **Scheme 1**, i.e., using the sorted pairwise index.
- The presented LSH **Scheme 2**, i.e., using the triplets index.
- The presented LSH **Scheme 3**, i.e., using the unsorted pairwise index.
- The presented LSH **Scheme 4**, i.e., using the unsorted pairwise index.
- The competitor **SimJoin**.

	NYT (MB)		Yago (MB)	
	$k = 10$	$k = 20$	$k = 10$	$k = 20$
Simple Inverted Index	14.99	22.08	15.99	30.18
Sorted Pairwise Index	64.81	267.45	107.0	437.59
Unsorted Pairwise Index	45.45	188.25	77.51	319.06
Triple Index	127.0	776.18	254.2	1042

Table 4.9: Comparison of size among indices

	NYT (sec)		Yago (sec)	
	$k = 10$	$k = 20$	$k = 10$	$k = 20$
Simple Inverted Index	0.081	0.114	0.075	0.095
Sorted Pairwise Index	0.342	1.732	0.368	1.886
Unsorted Pairwise Index	0.307	1.434	0.409	1.816
Triple Index	1.250	12.09	1.434	28.69

Table 4.10: Comparison of building time among indices

- We have additionally implemented **LinearScan** that is simply a full linear scan over all rankings. We do not report on it in the plots for readability, as **LinearScan** *is at least 34 times slower than* **InvIn**.

Before starting the evaluation of the performance of different proposed schemes for the similarity search, we present a comparison of the building time and the index size among different indices used in proposed LSH schemes in Table 5.3 and Table 4.10. From the statistics in these table, we can see that the size of the index grows as more elements are used to generate the keys, as expected. We can also find that the growth of this index increases more drastically for Yago data than the NYT, due to their data characteristics. As the elements in Yago rankings are less skewed, using multiple elements as keys does not shorten the size of posting lists compare to the case of NYT.

4.6.1 Evaluation of theoretical bounds on index access for Query-Driven LSH

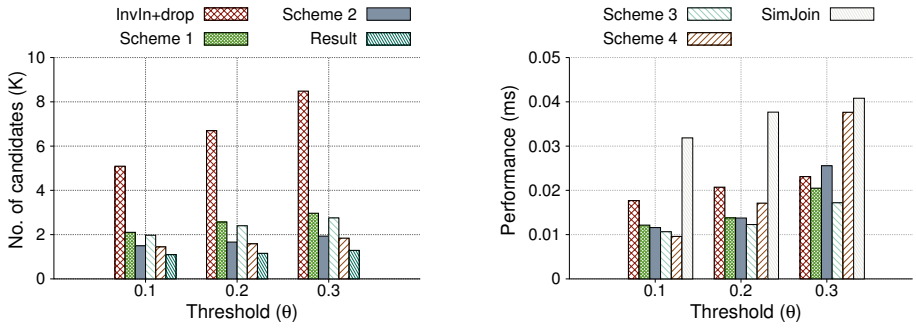
Table 4.11 and Table 4.12 tables present the theoretically established bounds for the parameter l, denoted as **TB**, determined from the Equation 4.3 - 4.6 for respectively Scheme 1, Scheme 2, Scheme 3, and Scheme 4 with error probability $\delta = 0.01$, confirming recall value 99%. In these tables, we compare the statistics under **TB** with the manually tuned minimum index accesses to achieve 100% recall in the similarity search, presented by column **M**. From Table 4.11 and Table 4.12, we can see that, almost all of the time, the automatically tuned parameter l is the same as the manually tuned l for 100% recall. For the cases where the manually tuned l is larger than the theoretically established one, the actual recall for this value of l is given within parentheses.

θ	Data	Scheme 1		Scheme 2		Scheme 3		Scheme 4	
		TB	M	TB	M	TB	M	TB	M
0.1	NYT	3	1	4	2	3	2	4	2
	Yago	3	3	4	3	3	3	4	3
0.2	NYT	5	4	8	6	5	5	8	6
	Yago	5	4	8	6	5	5	8	6
0.3	NYT	7	5	18	9	7	7	18	13
	Yago	7	6	18	11	7	7	18	15

Table 4.11: Comparison of tuning factor l for 100% recall for top-10 rankings

θ	Data	Scheme 1		Scheme 2		Scheme 3		Scheme 4	
		TB	M	TB	M	TB	M	TB	M
0.1	NYT	3	3	3	3	3	3	3	3
	Yago	3	3	3	3	3	3	3	3
0.2	NYT	6	6	8	8	6	6	8	8
	Yago	6(99.9)	7	8(99.8)	9	6(99.9)	7	8(99.8)	11
0.3	NYT	7	7	10	10	7	7	10	10
	Yago	7(99.9)	8	16	12	7(99.9)	9	16	12

Table 4.12: Comparison of tuning factor l for 100% recall for top-20 rankings

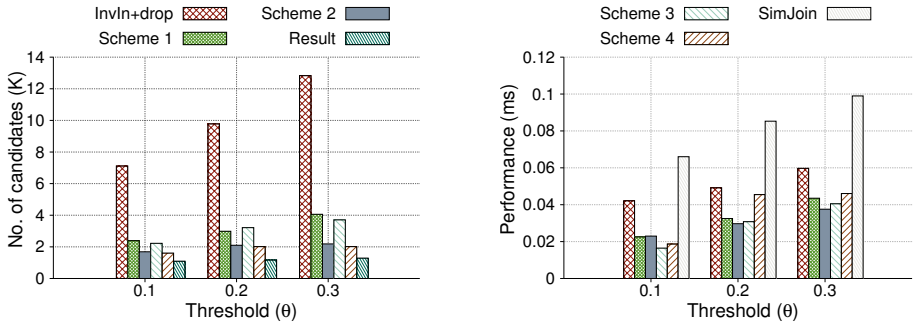


(a) $k = 10$ (b) $k = 10$

Figure 4.3: Comparative study of query processing for varying θ in top-10 Yago rankings.

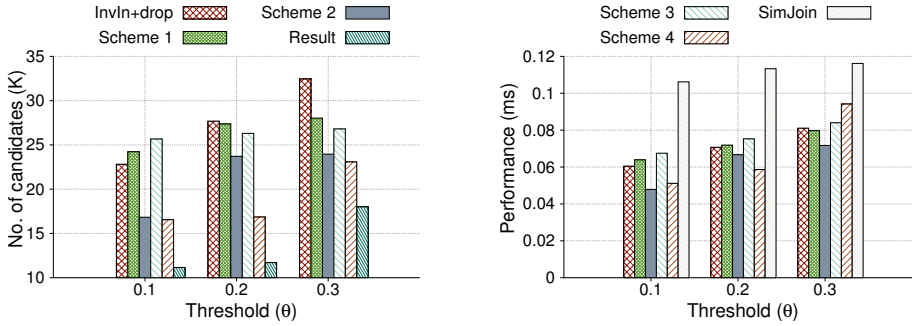
4.6.2 Performance Analysis among Query-Driven LSH Schemes and Competitors

The baseline approach InvIn retrieves the total of 22,755 and 37,071 candidates and takes on average 0.05ms and 0.15ms to response, respectively for top-10 and top-20 Yago rankings. InvIn retrieves the total of $60,627$ and $74,128$ candidates and takes on average 0.15ms and 0.35ms to response, respectively for top-10 and top-20 NYT rankings. **The performance of InvIn is more than three times slower for Yago and two times slower for NYT compared**



(a) $k = 20$ (b) $k = 20$

Figure 4.4: Comparative study of query processing for varying θ in top-20 Yago rankings.

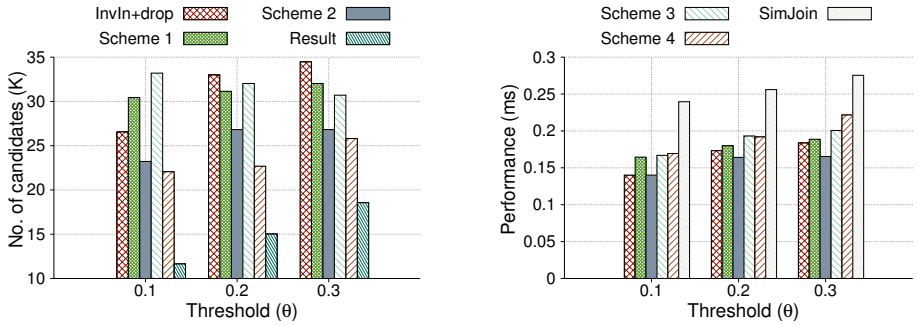


(a) $k = 10$ (b) $k = 10$

Figure 4.5: Comparative study of query processing for varying θ in top-10 NYT rankings.

to our proposed schemes. From Figure 4.4, we see that fewer candidates are retrieved from the pairwise indices or the triplets index using the proposed schemes, compared to the baseline and the InvIn+Drop method, for the Yago dataset. This performance improvement stems from the tight bound of the parameter l, resulting less index accesses, as well as from shorter posting lists due to the query-driven selection of hash tables. Since recall is tuned to 100%, retrieving fewer candidates implies evaluating fewer false-positive candidates, which reflects the characteristic of the LSH technique that true-positive candidates are more likely to be hashed into the same bucket. The characteristic of retrieving less false-positive candidates than the baseline and InvIn+Drop methods is consistently observed throughout the datasets—except for Scheme 1 and Scheme 3 with parameter $\theta = 0.1$ for the NYT dataset (cf., Figure 4.6).

In Figure 4.4 and Figure 4.6, we observe that retrieving less false-positive candidates for all the proposed schemes directly influences the runtime performance for both datasets. For all proposed schemes, as the threshold increases, more hash tables are needed to be accessed, and the runtime performance re-



(a) $k = 20$ (b) $k = 20$

Figure 4.6: Comparative study of query processing for varying θ in top-20 NYT rankings.

mains almost proportionate with the number of retrieved elements, for all approaches with both datasets. The only exception is observed for the Scheme 4 with $\theta = 0.3$, due to the computation time for finding intersection of posting lists from the key pairs used in Scheme 4, as discussed in Section 4.4.2. Figure 4.4 and Figure 4.6 also show that all the proposed schemes perform better than the competitor SimJoin. SimJoin finds the best prefix parameter for each query and applies Adapt-Join using a delta inverted index to find potential candidates. From the experimental results, for both datasets, we can conclude that the 1-prefix scheme performs best most of the time, following the observation that the InvIn+Drop method performs better than the competitor. More precisely, the InvIn+Drop scheme uses the 1-prefix method and ensures best performance by dropping the longest index entries. Unlike this, in the SimJoin method, the index entries for the elements are globally sorted by the inverted document frequency, and thus, dropping posting lists according to the adaptive best prefix parameter does not ensure the dropping of the large ones. For the Yago dataset, we can see that all the schemes perform better than InvIn+Drop (except Scheme 4 for top-10 rankings with $\theta = 0.3$). For the NYT dataset only Scheme 2 and Scheme 4 is winning over InvIn+Drop, except Scheme 4 with $\theta = 0.3$. One reason behind this observation is that the NYT dataset contains more skewed data which is further explained by a recall analysis among the schemes, using experimental results summarized in Table 4.13 and Table 4.14.

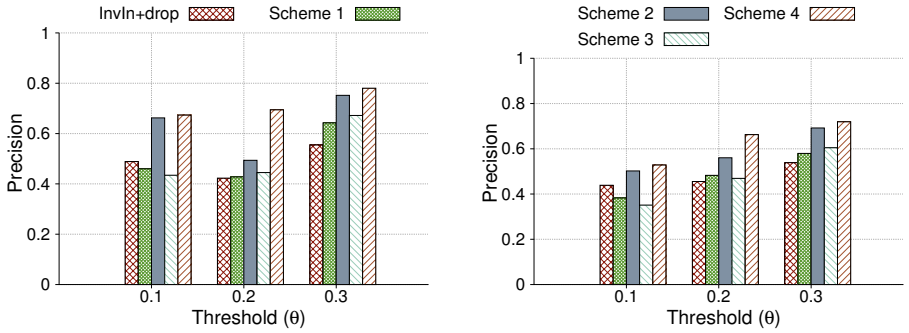
In Figures 4.4 and 4.6, we notice that Scheme 2 and Scheme 4 consistently retrieve fewer candidates than Scheme 1 and Scheme 3, respectively. A direct consequence of it is reflected in the precision studies; Figure 4.8 shows that the precision of Scheme 2 and Scheme 4 is always higher than Scheme 1 and Scheme 3. This result implies that the probability of retrieving candidates that belong to $\mathcal{R}$ in Scheme 2 and Scheme 4 is higher than in Scheme 1 and Scheme 3, respectively. This is align with the LSH property that the probability of finding similar ranking is higher when more hash functions are used to create hash keys for LSH schemes.

	$\theta = 0.1$				$\theta = 0.2$				$\theta = 0.3$			
	$l = 1$	$l = 3$	$l = 6$	$l = 10$	$l = 1$	$l = 3$	$l = 6$	$l = 10$	$l = 1$	$l = 3$	$l = 6$	$l = 10$
Scheme 1 for NYT	100	100	100	100	99.9	100	100	100	99.8	99.9	100	100
Scheme 2 for NYT	100	100	100	100	99.9	100	100	100	67.5	67.7	99.9	100
Scheme 3 for NYT	99.7	100	100	100	98.9	99.8	100	100	97.9	99.6	99.8	100
Scheme 4 for NYT	99.7	100	100	100	99.8	99.8	100	100	67.1	67.4	99.7	100
Scheme 1 for Yago	98.9	100	100	100	97.1	99.5	100	100	92.1	97.9	99.9	100
Scheme 2 for Yago	98.9	100	100	100	96.6	98.9	100	100	90.1	95.5	98.6	99.5
Scheme 3 for Yago	98.7	100	100	100	96.6	99.3	100	100	91.3	97.3	99.7	100
Scheme 4 for Yago	98.9	100	100	100	96.6	99.3	100	100	91.3	97.3	99.7	100

Table 4.13: Comparison of achieved recall in percent for $k = 10$.

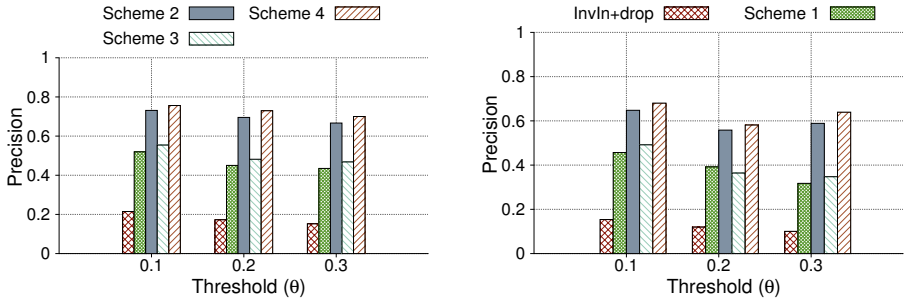
	$\theta = 0.1$				$\theta = 0.2$				$\theta = 0.3$				
	$l = 1$	$l = 3$	$l = 6$	$l = 10$	$l = 1$	$l = 3$	$l = 6$	$l = 10$	$l = 1$	$l = 3$	$l = 6$	$l = 10$	$l = 15$
Scheme 1 for NYT	100	100	100	100	99.9	100	100	100	99.2	99.9	100	100	100
Scheme 2 for NYT	99.9	100	100	100	99.8	99.9	100	100	85.4	85.6	100	100	100
Scheme 3 for NYT	99.7	100	100	100	98.6	99.5	99.9	100	97.5	99.2	99.8	100	100
Scheme 4 for NYT	99.8	100	100	100	98.5	99.7	99.8	100	83.5	85.0	86.2	100	100
Scheme 1 for Yago	99.1	100	100	100	96.4	98.8	99.9	100	92.0	96.8	99.2	99.6	100
Scheme 2 for Yago	99.1	100	100	100	95.6	98.0	99.6	100	90.5	95.2	98.4	99.6	100
Scheme 3 for Yago	99.0	100	100	100	95.6	98.4	99.5	99.8	90.8	96.3	98.9	99.5	100
Scheme 4 for Yago	98.8	100	100	100	94.7	97.7	98.9	100	89.1	94.9	98.2	100	100

Table 4.14: Comparison of achieved recall in percent for $k = 20$.



(a) $k = 10$, NYT (b) $k = 20$, NYT

Figure 4.7: Comparative study of precision for varying θ in NYT rankings.

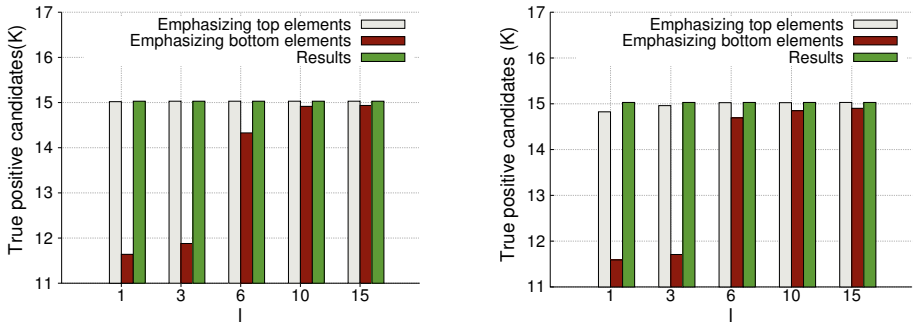


(a) $k = 10$, Yago (b) $k = 20$, Yago

Figure 4.8: Comparative study of precision for varying θ in Yago rankings.

Comparing the individual rows in Table 4.13 and Table 4.14, we can see that the recall achieved by Scheme 1 and Scheme 2 is always greater than the recall reached by Scheme 3 and Scheme 4 respectively, for the same value of l. This observation validates our theoretical analysis of recall bounds among the proposed schemes, shown in Figure 4.1 for different l. Putting it in another way, Scheme 3 is less likely to find a true-positive result than Scheme 1, for the same value of l. Additionally, comparing the columns of Table 4.13 and Table 4.14, we observe that the recall increases as l increases, which is in line with the LSH theory. We can also understand the characteristic of the datasets, by analyzing the recall statistics, shown in these tables. Comparing the rows of Table 4.13 and Table 4.14, for all the schemes with the same threshold θ and l, we find out that the recall for the NYT dataset is always larger or equal to the recall for the Yago dataset. This observation reflects that the elements of the NYT dataset are more skewed than in the Yago dataset.

Figure 4.8 presents the comparison by precisions of all four schemes. It shows that the precision of Scheme 2 and Scheme 4 is always higher than for Scheme 1 and Scheme 3, again, this is in line with the LSH theory. This observation also reflects that the NYT dataset is more skewed than Yago. Due to skewed



(a) Scheme 1, NYT (b) Scheme 3, NYT

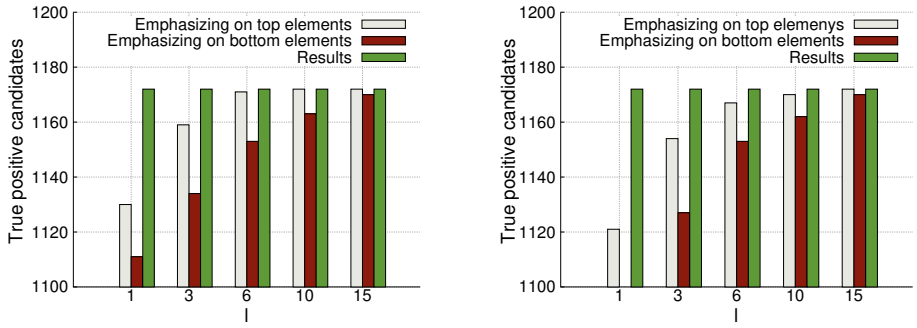
Figure 4.9: Comparative study of selection of hash functions for top-20 query with $\theta = 0.2$.

distribution of elements in the NYT dataset, the precision of all schemes for NYT remains almost the same as the precision of InvIn+Drop, whereas the precision of all four schemes for Yago data is much higher than the precision of InvIn+Drop.

4.6.3 Performance Analysis among Different At-Query-Time Selection Methods

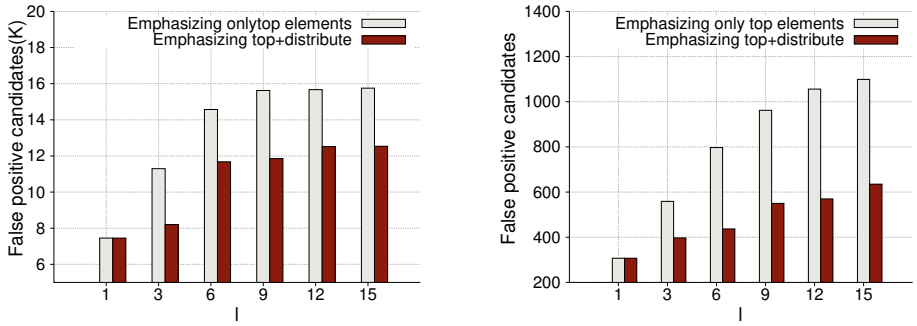
Figure 4.9 and Figure 4.10 report on the effect of the position of elements on selection of hash function for Scheme 1 and Scheme 3. From these figures, we can see that both schemes retrieve more true-positive candidates when hash functions are selected emphasizing on pairs or triplets from only top elements of the query instead of the bottom elements, for both datasets. We also see that the difference between retrieved true-positive candidates on focusing only the very top elements and the very bottom elements is larger in NYT, as the NYT data have more skewed distribution than Yago. For the same reason, 100% recall is also achieved faster by pruning false-positive candidates, shown in Figure 4.10. This property remains consistent to all other schemes.

Figure 4.11 shows that fewer false-positive candidates are retrieved using the selection method according to Algorithm 3 than the selection method according to the same algorithm when replacing Line 6 (i.e., to make the selection emphasizing only the top elements), for both datasets with the same value of l. This effect also remains consistent with other schemes. This result has direct influence in efficiency of similarity search as retrieving more false positive results need more validation time, in order to collect all results in $\mathcal{R}$. With the increasing ranking size, the efficiency suffers more because the validation time is proportionate with the ranking length. Considering Scheme 2, we need to access only eight entries in index for retrieving 100% results for given threshold 0.2



(a) Scheme 1, Yago (b) Scheme 3, Yago

Figure 4.10: Comparative study of selection of hash functions for top-20 query with $\theta = 0.2$.



(a) NYT (b) Yago

Figure 4.11: Effect of selection method for Scheme 2, $k = 20$ and $\theta = 0.2$.

(see Table 4.11) in top-20 list. However, 3772 more false positive candidates are retrieved as candidates from eight index scans while hash functions are selected only emphasizing top elements (see Figure 4.11). This difference becomes larger for more skewed dataset and it grows as l increases due to the generation of more duplicate candidates.

4.6.4 Lessons Learned

Now, we can summarize the main findings from the above experimental results as follows:

1. The value of the automatically tuned parameter l for 99% recall matches almost precisely with the actual value of l that is needed to retrieve all results.

2. Comparing the efficiency of similarity search among all proposed schemes, Scheme 2 performs best for skewed data (e.g., NYT) whereas Scheme 3 performs best for less-skewed dataset.

3. There is a tradeoff between the space needed to store the index structure for the proposed schemes and the efficiency during similarity search. Though Scheme 3 does not perform best for a skewed dataset, it uses less space to store the hash tables (i.e., the unsorted pairwise index) compared to the space needed to store the hash tables for Scheme 2 (i.e., the triple index).

4. All the schemes outperform the adaptive prefix-filtering method for similarity search and the manually tuned optimal baseline approach InvIn+Drop.

4.7 Summary

In this chapter, we presented an efficient approach to processing similarity queries over top-k lists under the generalized Kendall's Tau distance. We proposed four different LSH schemes using two different hash function families, reflecting different ways to realize and understand pair-based and triplet-based indices. Key contribution of this work is the in-depth analysis of the ability of the proposed methods to determine high-recall results with few lookups in the index. We have derived query-driven formulations of the expected recall and presented bounds that allow an automated tuning of the number of hash tables required to achieve a predefined recall. We implemented the described approaches and reported on the results of a comprehensive performance evaluation, using two real-world datasets. This study confirmed the insights obtained through theoretic analysis and further demonstrated the superiority of the approaches over plain inverted indices and SimJoin, the state-of-the-art from literature.

Chapter 5

LSH-Based Probabilistic Pruning of Inverted Indices

This chapter is based on our own publication at WebDB 2017 [PM17]. In this chapter, we aim at optimizing the usage of hash tables for our proposed LSH methods, which implies pruning of the entries in pairwise indices or triple index, discussed in the previous chapter.

In Section 3.2, we briefly mentioned query-aware LSH methods [JASG08, HFZ+15] that suffer from a similar problem of keeping most of the hash tables, though only a few of them are used during query processing. On the other hand, the methods used in literature such as LSH forest [BCG05] or locality sensitive B-tree structure [TYSK10] are not applicable in our proposed LSH schemes. Because the proposed schemes in the previous chapter use up to three hash functions together (in case of Scheme 2) and also the hash functions map data to the binary space, which leaves not much choice for building a prefix tree or B-tree over the labels of hash buckets.

In this chapter, we present a method for optimizing the number of hash entries required to maintain without reducing the quality of the similarity search over rankings by exploiting the locality sensitivity property of proposed hash families $\mathcal{H}_1$ and $\mathcal{H}_2$. So, the objective is to prune the inverted indices in a way such that it does not effect the result quality. Here, we formulate the problem statement in a generic way, relaxing the space optimization problem in similarity search over rankings as well as similarity search over sets. We will, throughout the coming sections, focus on sets, but come back to handling ranked lists in Section 5.5, where we instantiate the proposed "pruning approach" for two explicit scenarios.

Essentially, all inverted indices resemble a hash map where the keys are items, pairs, etc., derived from the sets/rankings to be indexed. In the previous chapter, we have seen how such indices can efficiently determine very similar sets/rankings by accessing fewer index entries than the prefix-filtering method,

$\tau_1 = \{2, 5, 4, 3, 1\}$	$7 \rightarrow \langle\tau_2\rangle, \langle\tau_3\rangle$
$\tau_2 = \{1, 4, 7, 5, 2\}$	$5 \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle, \langle\tau_3\rangle$
$\tau_3 = \{0, 8, 7, 5, 6\}$	$4 \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$
	...

Table 5.1: Sample sets (left) and inverted index (right)

by exploiting the LSH property. Let us emphasize briefly the connection between LSH and inverted indices through an small example. Table 5.1 shows three sets of integer values. The connection between inverted indices and locality sensitive hashing (LSH) is apparent: We can use the definition of $\mathcal{H}_1$ that projects sets τ_i to binary space based on presence or absence of the elements inside it. According to $\mathcal{H}_1$, the bucket label '1' for function h_2 is similar to the key entry '2' in the inverted index, both of them hold the sets where number '2' appears. In this work, we use the term 'hash bucket' and 'index entry' interchangeably.

This work also deals with the problem of similarity search over sets, as mention earlier, where the query size is the same as the size of the stored sets. Whereas, in score-based document retrieval method, queries are normally very short compared to the size of stored documents. Hence, finding the results based on score-based document search differs from the searching scenario considered in our work. Similarly, let us further briefly mention that there are many lossless compression methods and lossy static index pruning methods for web search engines and document retrieval in general [SCC+01, ZM06, SJPB08]. We believe that lossless compression techniques are fully orthogonal to our approach. We will discuss how the static pruning methods can be related to our proposed pruning approach later in Section 5.3.

5.1 Problem Statement and Setup

Consider a data collection $\mathcal{S}$ comprising of sets τ_i. Each $\tau_i \in \mathcal{S}$ has a domain D_{τ_i} of items it contains. The global domain of items is then $D = \bigcup_{\tau_i \in \mathcal{S}} D_{\tau_i}$ and $|D| = n$. We assume all sets have the same size. Table 5.1 shows three sets holding five elements each.

At query time, a user provides a query set q, a distance threshold $\theta \in [0, 1]$, and a distance function d. Our objective is to determine all sets $\tau_i \in \mathcal{S}$ that have a distance less than or equal to θ, so the result $\mathcal{R}$ of a query can be written as:

$$\mathcal{R} := \{\tau_i \,|\, d(\tau_i, q) \leq \theta, \tau_i \in \mathcal{S}\}.$$

As mentioned above, we can build a simple inverted index over $\mathcal{S}$, to look up—at query time—those sets that have at least one element overlapping with the query's elements. Particularly, the distance to the query is evaluated only for the sets which hold overlapping elements with the query (c.f., the Filter and Validate method presented in Algorithm 1 in Section 2.1.2). Considering the

example in Table 5.1, for a query $q = \{8, 7, 0, 6, 9\}$, the set τ_1 does not overlap at all with the query's elements, while τ_2 and τ_3 do overlap.

In this chapter, we address the problem of rendering inverted indices more space-efficient by eliminating some of the stored posting lists, posting list entries, or both, controlled by a pruning parameter $\phi \in [0, 1]$.

If a query is executed over such a pruned index, the query returns a result set $\mathcal{R}_p$ that is a subset of the true result set $\mathcal{R}$ given above.

Our objective is to find the optimal pruning factor ϕ such that the results of the similarity search satisfy a user-defined recall level ϱ. We can formalize this task as follows.

$$\begin{aligned} &\textit{maximize} \quad \phi \\ &\text{subject to} \quad \mathcal{R}_p/\mathcal{R} \geq \varrho \end{aligned} \tag{5.1}$$

Note that we assume the distance threshold θ to be strictly smaller than the normalized maximum possible distance, hence, $0 \leq \theta < 1$. Therefore, the inverted index can find all result sets as all results need to have at least one overlapping item with the query.

5.2 Contributions and Outline

In this work, we make the following contributions:

- We discuss three ways to prune an inverted index and propose two different ways to query the pruned index.
- We present a probabilistic analysis to find the number of index entries needed to be accessed to ensure the predefined recall goal for the similarity search over pruned inverted indices.
- Based on the analysis, we formalized an optimization problem to find maximum pruning factor ϕ^*, while ensuring the user-given recall goal.

The remainder of this chapter is organized as follows. Section 5.3 discusses the three, admittedly very straightforward, pruning techniques. The probabilistic analysis of query processing in pruned indices and optimization problem of finding the maximum pruning factor are proposed in Section 5.4. Section 5.5 discusses two case studies that apply our proposed techniques to prune and query inverted indices for sets and top-k rankings. Section 6.4 presents a detailed experimental study and finally, Section 5.7 summarizes the work of this chapter.

5.3 Horizontal, Vertical, and Diagonal Index Pruning

As shown in Section 4.3 of the previous chapter, the space complexity of pairwise indices or triple index grows polynomially with the number of elements used as keys. Subsequently, optimizing the space requirement of such indices becomes the necessity as it is important to keep the index in main memory to avoid expensive random I/O while accessing the hash buckets. In this section, we discuss three simple ways to drop parts of the inverted index structure, as illustrated in Figure 5.1.

5.3.1 Horizontal Pruning

In this approach, we randomly select a fraction ϕ of the total index keys and remove the corresponding posting lists completely. Considering the elements are uniformly distributed over hash buckets, elimination of random posting lists reduces ϕ fraction of space required to store complete index. Removing random index keys signifies removing random hash functions from the hash family in LSH, and thus, discarding few hash entries that are equivalent to the pairwise or triplet-based indices.

On the other hand, instead of removing the posting lists completely, we can drop the most frequent or least frequent ϕ fraction of index keys (thus, drop very long or very short posting lists). In this case, we can calculate the reduced space due to the removal of posting lists based on the skewness of the data. For example, having data following a Zipf distribution with skewness parameter $\nu = 1.1$ implies that removing top-20 entries of index keys will remove 30% of total objects listed in all posting lists. Clearly, random removal of a ϕ fraction of entries reduces less space than the removal of the top ϕ fraction of entries and more space than the removal of the bottom ϕ fraction of entries.

5.3.2 Vertical Pruning

Unlike removing index keys and the associated entire posting lists, in this strategy, we randomly drop a ϕ fraction of items from each posting list. This is the scenario when we keep all hash tables from the hash families but remove ϕ fraction of items from each hash buckets. Eliminating entries from hash buckets inevitably leads to removing items from the equivalent entries in the pairwise or triplet-based indices. The space reduction due to this pruning approach does not depend on the distribution of the elements. Dropping a ϕ fraction of items from each posting lists always reduces the space requirement to store posting lists by a factor of $(1 - \phi)$.

Assuming items from a posting list contain information about the usefulness of them, we can drop the least important ones from each list, instead of randomly dropping a few items. In information retrieval literature, such an approach is

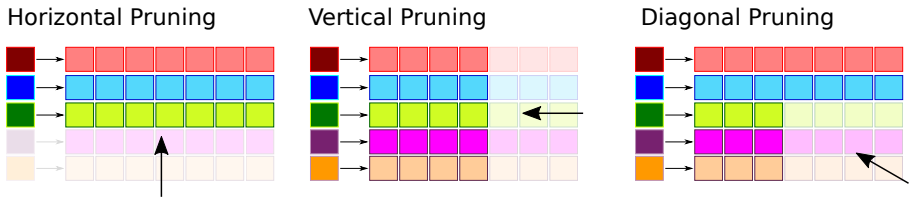


Figure 5.1: Illustration of three ways to prune an inverted index

known as term-based static index pruning [SCC+01, dMdSF+05], where the authors use different scoring models to decide the importance of an item in the posting list.

5.3.3 Diagonal Pruning

In this strategy, we drop a fraction of ϕ elements from each of the original sets and then create an index based on the remaining elements. Hence, if a specific element is, by chance, removed from all the objects, the index entry corresponding to that removed element will not appear in the index, which resembles the idea of horizontal pruning. On the other hand, if a specific element is removed only from few sets, then those sets will not appear in the posting list of that element, which resembles vertical pruning. This is the scenario when, for each object, we randomly pick ϕ fraction of hash functions from the hash family to map the object. Thus, if a hash function is never chosen for mapping any of the objects, the entire hash table for that hash function is lost, i.e., the equivalent index keys in the pairwise or triple index for that hash function is also lost, yielding the horizontal pruning scenario. On the other hand, if a hash function is chosen occasionally for mapping few of the objects, some of the entries from that hash table are eliminated, causing vertical pruning of equivalent pairwise or triple index entries.

As this method prunes the index both horizontally and vertically, we called it diagonal pruning. It generalizes the document-centric pruning method [BC06], known from the area of information retrieval, where less important terms are removed from each document before generating the index. We can observe here that if elements are uniformly distributed among sets, this pruning method tends to prune the index equally from both vertical and horizontal direction. On the other hand, if data has a skewed distribution, then this method tends to prune the index more horizontally than vertically.

For all three pruning methods, discussed above, it is clear that eliminating an object from the index affects the query processing. In the next section, we will discuss how we can quantify the pruning effect on the quality of the search results in similarity search.

5.4 Query Processing on Pruned Indices

Due to pruning of an element from the index, i.e., pruning of an element from the hash bucket, we can miss a candidate for a query if the query and the pruned element are supposed to collide into the same bucket by a hash function. Thus, pruning increases the chance of missing a true result in similarity search. In case of dropping an entire posting list corresponding to a key, a collision can be missed if the hash function maps the query to the pruned bucket. As we can relate the inverted index with the hash tables of a hash family, discussed in the previous chapter, the probability of missing result can be approximated by exploiting the modification of the collision probability of hash families. Here, we discuss two different ways to query a pruned index exploiting LSH such that the predefined recall requirement can be ensured.

5.4.1 Ad-hoc Query Processing

A properly set up LSH ensures that l accesses to the hash tables collect enough candidates to achieve predefined recall goal. But, while dropping hash entries or elements from hash buckets, which is also similar to querying a pruned index, as discussed in the previous section, the assurance of result quality does not hold anymore. One simple solution to overcome this problem is using more than l hash functions to map the query into the pruned index, until we reach l successful accesses to hash buckets. In the worst case, we might need to use many hash functions to map the query into the pruned index, in order to achieve the required number of successful accesses. In this scenario, a missed collision, i.e., the case when the hashed query does not find any matched key in the pruned index, does not collect any additional candidates for that index access. As the runtime performance is mainly dominated by the validation of retrieved candidates (i.e., actual distance computations between candidates and the query) from successful bucket accesses, this approach does not incur any extra overhead in the runtime performance.

5.4.2 Probabilistic Query Processing

Unlike the above ad-hoc approach, in probabilistic query processing, we modify the collision probability of LSH schemes based on the fraction of objects pruned from the index and the pruning methods (i.e., horizontal, vertical, or diagonal). Then, based on the modified collision probability, we calculate new bound on the number of required accesses to the pruned index, in order to assure the user-defined recall.

Consider a factor f_Y that scales the original collision probability P_1 according to the fraction of pruned object. Introducing the factor f_Y in recall Equation 2.1, discussed in Section 2.1.3, we get Equation 5.2, which is further used to tune the old parameter l for a pruned index. We find the the number of required accesses l_Y by solving Equation 5.2 that adapts the effect of pruning in query

processing and assures the user-defined recall.

$$\varrho = 1 - (1 - f_Y \cdot P_1^m)^{l_Y} \tag{5.2}$$

In notation of l_Y, the subscript Y indicates the pruning method, i.e., horizontal, vertical, or diagonal, that was used for pruning the index. Required accesses l_Y refers to the number of accesses in the pruned index (hash buckets) that are needed in order to reach a predefined recall goal under the *modified collision probability*. Now, we will discuss how to calculate f_Y for all three pruning methods.

Horizontal pruning: Consider the items are uniformly distributed over a collection of sets $\mathcal{S}$. Therefore, when building an inverted index over $\mathcal{S}$, the sets will be evenly distributed over each index entry, i.e., each posting list will hold almost similar number of sets. Therefore, removing ϕ fraction of index keys randomly from the index will prune total ϕ fraction of postings (cf., sets) from the index. Clearly, a query can not collide with a candidate set, which is removed due to pruning of the index key, where the query is mapped by a hash function.

Hence, the collision probability P_1 is modified by the factor $f_h = (1 - \phi)$, as a collision can occur only for non-pruned index entries. For a non-uniform distribution of the sets into the buckets of the index, the pruning factor can be modified using, if known, the cumulative distribution function (CDF) of the pruned objects, i.e., the total number of posting elements which are removed due to the pruning of ϕ fraction of index entries.

Vertical pruning: Similar to horizontal pruning, the collision probability P_1 is modified by the factor $f_v = (1 - \phi)$ for the vertical pruning. Although the keys in the index are not pruned, the collision between objects can still be missed due to removal of ϕ fraction of postings from each index entry (i.e., posting list). Considering l_h is the number of keys that we need to look up to reach the predefined recall goal ϱ with modified collision probability, we have $l_h = l_v$. Note that l_h is equal to l_v only when we consider that the data (cf., sets) are uniformly distributed over index entries.

Diagonal pruning: In diagonal pruning, we prune ϕ fraction of items from each set before creating the inverted index, i.e., a specific set will be listed under posting lists of only $1-\phi$ fraction of items from that set. In this pruning method, a query may not collide with a candidate set due to the following two reasons:

- The items in the index key are always chosen from pruned items of the sets, i.e., the hash function that maps both the query and the set into the same bucket was never chosen during hash table creation.

- The items in the index key are chosen from pruned items for few sets, including the candidate set, but not for all, i.e., the hash function that maps both the query and the set into the same bucket was selectively not used during the mapping of the candidate set.

Therefore, we need to find the number of postings (cf. sets) removed from the posting lists for both the reasons, mentioned above. Here, it is important to know how the keys are generated to build the inverted index. Considering i items are used to generate the keys, the pruning of ϕ fraction of items from each set prunes $1 - \binom{k'}{i}/\binom{k}{i}$ (with $k' = (1-\phi) \cdot k$) fraction of postings from the index because each object is listed under only $\binom{k'}{i}$ non-pruned keys. Consequently, the collision probability P_1 is adjusted by the following factor:

$$f_d = \frac{k'(k'-1)\cdots(k'-i-1)}{k(k-1)\cdots(k-i-1)}.$$

5.4.3 Cost Model for Query Processing

Query processing is mainly divided into two parts: In the first step, the hash tables are accessed to retrieve candidates (filter step). Then, all candidates are validated to determine the result set $\mathcal{R}$.

The cost of the filtering step depends on the number of index accesses at query time and the length of the posting list. Based on the distribution of elements in the collection of sets S, we can find the expected length of posting lists, denoted as $Post_{len}$. In *probabilistic query processing*, we can find the number of required accesses for different pruning methods, i.e., l_v, l_h, and l_d from Equation 5.2, by plugging in the specific factors f_Y discussed in the previous section.

On the other hand, the required index accesses for *ad-hoc query processing* depends on the number of successful scans, that means, accesses via keys that actually exist in the pruned index. The probability of a key being present in a pruned index is equal to the factor f_Y, discussed in Section 5.4.2, depending on the pruning methods. Considering each index access as a *Bernoulli trial*, we can calculate the number of trials to get the first successful access. Therefore, we calculate the expected number of scans, $E[l_v]$, $E[l_h]$, and $E[l_d]$ for ad-hoc query processing which leads to l successful accesses for vertical, horizontal, and diagonal pruning, respectively, by the following equation:

$$E[l_Y] = l \cdot (1/f_Y). \tag{5.3}$$

Finally, to assess the cost of the validation phase, we need to find the number of candidates retrieved from filtering step, i.e., the union of the retrieved elements from accessing l_Y entries from the hash tables. Considering $|D| = n$, i.e., a total of n items that are used to build the index, the expected number of unique candidates is $E_{can}[l_Y] := n(1 - (1 - Post_{len}/n)^{l_Y})$. Let us consider c_f and c_v are the scanning cost and the cost for the distance calculation for candidate validation, respectively. So, the final cost to find $\mathcal{R}$ is given by:

$$Cost(l_Y) = (l_Y \cdot Post_{len} \cdot c_f) + (E_{can}[l_Y] \cdot c_v). \tag{5.4}$$

5.4.4 Optimizing the Pruning Factor

Now, we discuss how we use the above cost model to determine the optimal value of the pruning parameter ϕ. In Section 5.1, we have already defined the optimization problem, subject to a recall guarantee ϱ. Based on different pruning methods, we can estimate the expected number of index accesses (i.e., l_v, l_h, or l_d) that are needed to ensure the recall requirement, as discussed earlier. Following our discussion, if the pruning factor increases, the number of accesses in the pruned index also increases. However, there is a limit on the number of keys that are possible to generate from a query. Therefore, if the required l_Y accesses is over this limit, i.e., we cannot create l_Y number of keys from a query, the recall requirement cannot be ensured. Considering the length of each set in $\mathcal{S}$ is k and index keys are generated using i items from a set, we can generate maximum $\binom{k}{i}$ keys from a size-k set or from a query (for instance, $i = 2$ for pairwise index, discussed in Section 4.3.1). Hence, to ensure the constraint in Equation 5.1, we formulate the optimization problem as follows:

$$\begin{aligned} &maximize \quad \phi \\ &\text{subject to} \quad \tbinom{k}{q} - l_Y = 0, \\ &\qquad\qquad Cost(l_Y) \leq \beta \cdot Cost(l) \end{aligned} \tag{5.5}$$

As the required number of index accesses is directly proportionate to the pruning factor, i.e., we need to access the index more often if we prune the index more, we add an additional constraint in Equation 5.5, by restricting the increasing cost of query processing. To do so, we introduce a parameter β in Equation 5.5, where $\beta \geq 1$, based on the application scenario and the user demand.

In our thesis, we mainly deal with top-k rankings and generalized Kendall's Tau as the distance function. Hence, the validation cost c_v, i.e., the cost for distance computation for each candidate is $O(k \log k)$, whereas index access cost c_f is only $O(1)$. Clearly, $c_f \ll c_v$ depending on the size of the object (i.e., k). As a result, in our work, the candidate validation cost dominates the cost of a query processing and the filtering cost becomes negligible in the cost model. We also observe that the increasing number of index accesses in the pruned index does not affect the total number of unique candidates retrieved from a pruned index. Therefore, we can relax the bound of the parameter β by $\beta > 0$. With this new bound on β, the cost constraint looses its significance as the total number of unique candidates retrieved from a pruned index, as well as from a non-pruned index, is bounded by $O(|\mathcal{R}|)$. Therefore, in our work, the optimization problem can be simplified by removing the cost restriction and the optimal pruning factor ϕ^* is calculated as follows:

$$\phi^* = argmax_\phi \left\{ \tbinom{k}{q} - l_Y = 0 \right\}. \tag{5.6}$$

5.5 Case Studies

To demonstrate the versatility of our approach, we present here similarity search problem with two different distance measures. In the first scenario, we consider the similarity search over sets using the Jaccard distance, discussed in Section 2.1.4. The second scenario deals with the problem addressed in this thesis, i.e., the similarity search over rankings using the generalized Kendall's Tau distance. In this section, we discuss different parameters that are used to find the optimal pruning factor for each scenario.

5.5.1 Scenario I: Set Similarity with Jaccard Distance

In general, we can use a single-item-based inverted index to determine the candidate sets that overlap at least one item with the given query set by measuring the number of overlap between them. As mentioned earlier, with the example given in the introduction of this chapter, we can simply use proposed hash family $\mathcal{H}_1$ to relate LSH to querying single-item-based inverted index. Hence, $\mathcal{H}_1$, proposed in Section 4.4.1, is the appropriate choice of hash family as the order of elements in the set is not required to find the overlap-based Jaccard distance. Moreover, using an index that combines two elements together as key, we can find the candidates that are more similar to the query. Therefore, we want to build here a sorted pairwise index presented in Section 4.3, based on the pairs of elements from each set. For example, one such index entry of the sorted pairwise index for the example shown in Table 5.1 looks like $(4, 5) \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$. Following the discussion how the sorted pairwise index is directly related to the index used by the proposed query-driven LSH method Scheme 1, defined in Section 4.4.1, we apply our pruning approach on a sorted pairwise index by exploiting Scheme 1.

According to the query-driven LSH method, Scheme 1 uses the collision probability $P_1 = \mu/k$, where μ *is the minimum overlap needed to meet a distance threshold* θ. In the previous chapter, we define μ corresponding to the generalized Kendall's Tau distance. Hence, we need to find the parameter μ for the Jaccard distance here. To meet the Jaccard distance threshold θ, we need at-least θ_s Jaccard similarity, where $\theta_{=}1-\theta$. Now, considering μ is the minimum overlap needed to meet a Jaccard distance threshold θ, the required Jaccard similarity is $\mu/(2k-\mu)$. Hence, by solving $\theta_s = \mu/(2k-\mu)$, we find $\mu = 2k\theta_s/(1+\theta_s)$.

For Scheme 1, the parameter $m = 2$, discussed in Section 4.4.1. We can now apply m, μ, and set size k in Equation 4.3 to determine the required number of accesses l in complete index for a fixed recall goal. Further, using the same parameters, additionally with f_Y in Equation 5.2, we can find the required number of accesses l_Y in pruned index for a fixed recall goal. Clearly, we need to consider $i = 2$ to find modifying factor f_d for the diagonal pruning method and in Equation 5.6—as two items are used to create keys.

5.5.2 Scenario II: Ranking similarity over Kendall's Tau Distance.

Similarity search over a collection of rankings under generalized Kendall's Tau distance has been discussed in the previous chapter, by using different variations of inverted indices. In the previous scenario, we applied our pruning approach to one of these inverted indices, the sorted pairwise index, by exploiting LSH family $\mathcal{H}_1$. Hence, in this scenario, we consider another variation of pairwise indices, the unsorted pairwise index, presented in Section 4.3. This index also uses pairs of items as keys, but there is a difference between the key (i, j) and the key (j, i): The posting list for the key (i, j) holds only those rankings where element i occurs before j, and vice versa for the key (j, i). For instance, entry $(4, 5)$ only holds τ_2 but not τ_1 for the example shown in Table 5.1. In the previous chapter, we have discussed that the unsorted pairwise index is directly related to the index used by LSH method Scheme 3, defined in Section 4.4.2. Moreover, Scheme 3 is defined based on the LSH family $\mathcal{H}_2$, proposed in Section 4.4.2 and it is the most efficient one for this scenario, discussed in Section 4.6.4. Therefore, in this scenario, we apply our pruning approach on the unsorted pairwise index by exploiting Scheme 3.

Here, we directly obtain the collision probability P_1, m, and l from the discussion in Section 4.5.1. Using the same parameters, additionally with f_Y in Equation 5.2, we can find the required number of accesses l_Y in pruned unsorted pairwise index for a fixed recall goal. Similar to the above scenario, the diagonal pruning method considers $i = 2$ to find f_d and then, Equation 5.6 is used to determine the optimal pruning factor.

5.6 Experimental Evaluation

We have implemented the pairwise indices mentioned above in Java 1.7 and run the experiments using an Intel Core i7@3 GHz machine with 8GB main memory. Both pairwise indices used in experiments are kept entirely in memory.

The querying performance is given in terms of query response time (here, **wallclock time**), **number of retrieved candidates**, and **recall**. The results are provided by averaging five consecutive experimental runs over 1000 benchmark queries. We use two different datasets for the experiments.

LiveJ: A dataset containing a set of user profiles, describing the interests of users, is obtained from Life Journal[1]. This dataset is used for **Scenario I**. We consider 100,000 profiles of fixed (truncated) size 20 for the experiments.

Yago Entity Rankings: This dataset is used for the evaluation of **Scenario II**. From this dataset, we consider 25,000 top-20 rankings that have been mined from the Yago knowledge base, as described in [IMS13].

In this section, we discuss the effect of pruning methods in similarity search

[1] `http://socialnetworks.mpi-sws.org/data-imc2007.html`

Methods	θ for LiveJ (Scenario I) 0.1	0.3	0.5	θ for Yago (Scenario II) 0.1	0.2	0.3
Full scan	52006.1	52006.1	52006.1	37.21	37.21	37.21
Prefix-filtering	23185.1	34045.4	43988.1	19.26	22.78	25.95
Baseline	5105.3	7360.4	9059.5	2.327	2.706	3.067

Table 5.2: Retrieved candidates for different approaches of query processing in non-pruned index.

Pruning method	LiveJ θ	ϕ^*	l_Y	Yago θ	ϕ^*	l_Y
Horizontal pruning	0.1	0.8	125	0.1	0.9	53
	0.3	0.8	167	0.2	0.9	68
	0.5	0.7	112	0.3	0.9	90
Vertical pruning	0.1	0.8	125	0.1	0.9	53
	0.3	0.8	167	0.2	0.9	68
	0.5	0.7	112	0.3	0.9	90
Diagonal pruning	0.1	0.5	95	0.1	0.7	73
	0.3	0.5	126	0.2	0.7	97
	0.5	0.4	87	0.3	0.7	130

Table 5.3: Theoretically established optimal pruning factor ϕ^*.

based on experimental results for both scenarios, mentioned in Section 5.5, to validate our optimization problem. The effect of pruning methods is presented against the baseline approach. *The query-driven LSH method on the non-pruned index structures is considered as the baseline approach here.* Before starting the evaluation on pruned indices, we present the performance of the full scan and prefix-filtering method in the single-item-based index in Table 5.2. Based on Proposition 4.2.2, we determine the prefix-filtering parameter for both scenarios, where $\mu = 2k\theta_s/(1+\theta_s)$ as discussed in Section 5.5 for the first scenario, and $\mu = \lceil k(1-\sqrt{\theta})\rceil$ as discussed in Lemma 4.2.1 for the second scenario. The results show that both approaches retrieve far more candidates (> 5 times) than the baseline approach, as expected, for both scenarios.

5.6.1 Theoretically Established Parameters

In this experimental studies, both pairwise indices are pruned randomly using all three pruning methods. Based on Equation 5.6, Table 5.3 represents the theoretically established optimal pruning factor and corresponding number of index accesses l_Y, for predefined recall goal $\varrho = 0.99$, for both datasets. We should mention here that we consider discrete values of ϕ with step of 0.1, while optimizing ϕ^* using Equation 5.6. Due to randomly pruning of the index structures, we have $l_v = l_h$, as discussed earlier. Hence, the maximum pruning factor also becomes the same for both methods.

	Yago			LiveJ		
θ	0.1	0.2	0.3	0.1	0.3	0.5
l	3	5	7	2	4	8
$E[l_v]$	30	50	70	10	20	26.6
$E[l_h]$	30	50	70	10	20	26.6
$E[l_d]$	42.6	71	99.4	8.6	17.39	23.52

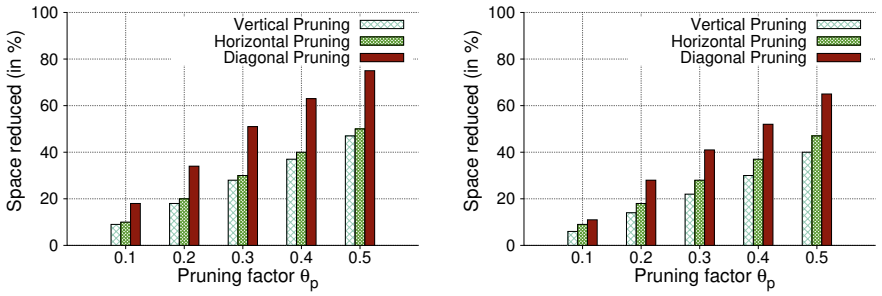
Table 5.4: $E[l_Y]$ for l successful accesses

Based on Equation 4.3 and Equation 4.5, we present the required number of accesses l for the sorted pairwise index (used in LiveJ) and the unsorted pairwise index (used in Yago), respectively, in Table 5.4. Additionally, in this table, based on Equation 5.3, we present theoretically established required number of access l_Y for ad-hoc query processing, in order to meet l successful scan in pruned indices, for optimal pruning factor presented in Table 5.3.

5.6.2 Efficiency of Pruned Indices

In this section, we first show the effect of index pruning with varying θ_p for both datasets in Figure 5.2, while discarded index keys are chosen randomly. For Yago data, we can see that both the horizontal pruning and vertical pruning method prune the size of the unsorted pairwise index almost proportionally to the pruning factor. On the other hand, the diagonal pruning method prunes the posting lists in proportion close to the factor of $2\theta_p$. This is expected as removed keys or removed objects from index are chosen randomly and also the elements in Yago rankings are less skewed, discussed in Section 4.6 of the previous chapter. From Figure 5.2, we can observe that all three pruning methods prune less amount of index space for a specific pruning factor in case of LiveJ data than Yago dataset, specially, for horizontal and diagonal pruning. The reason behind this characteristic is that the distribution of elements in profiles of LiveJ is more skewed than Yago. The length of posting lists in LiveJ lies in $[1, 19335]$, whereas The length of posting lists in Yago lies in $[1, 74]$. Due to the skewed distribution of data in LiveJ, the difference between horizontal and vertical pruning is also prominent, and during diagonal pruning, index entries are pruned more vertically than horizontally.

Using the theoretically established optimal pruning factor ϕ^*, we first prune the sorted pairwise index that is used in Scenario I. Then, we perform the proposed query processing methods for different θ over pruned index, and present the results in Table 5.5. Similarly, we perform the query processing on pruned unsorted pairwise index for Scenario II, and present the results in Table 5.6. From both tables, we can observe that the theoretically established ϕ^* fulfills the recall requirement ϱ for probabilistic query processing, and thus, validates the optimization problem. We can also notice that the probabilistic query processing retrieves more candidates than the baseline for the LiveJ dataset due to the skewed data distribution over index entries.



(a) **Yago** (b) **LiveJ**

Figure 5.2: Pruning effect in size of pruned index.

On the other hand, due to less skewed distribution of the items in the Yago dataset, the difference between retrieved candidates for the optimally pruned index and non-pruned index is negligible, which validates our assumption to discarding the cost factor to simplify the optimization problem. From the experimental results, we find that the runtime is proportional to the retrieved candidates that also support the simplification of the optimization problem.

Table 5.5 and Table 5.6 also report on the number of successful index accesses at query time, which is bounded by l_Y and $E[l_Y]$ in probabilistic and ad-hoc query processing, respectively. From the highlighted column in Table 5.4, Table 5.6, and Table 5.5, we observe that the $E[l_Y]$ is indeed match with the experimental results for horizontal pruning and can ensure the recall goal.

In Table 5.5 and Table 5.6, experimental results show that the ad-hoc query processing always retrieves less candidates than the probabilistic query processing and do not ensure the required recall requirement due to its ad-hoc characteristic. However, this method reaches the recall requirement only when #total accesses in ad-hoc query processing becomes almost the same as the expected number of index accesses given in Table 5.4. It also validates/supports our probabilistic analysis for ad-hoc query processing. For example, in horizontal pruning for both scenarios, ad-hoc query processing reaches recall requirement ϱ and we can see that #total accesses $\approx E[l_h]$ (highlighted in blue). As vertical pruning does not discard any keys from the index, the #total accesses in ad-hoc query processing is the same as l in baseline method, which is far less than $E[l_h]$. Therefore, ad-hoc query processing can not reach the recall requirement while using vertical pruning.

5.6.3 Lessons Learned

Here, we summarize the findings obtained through the experimental studies.

- The theoretically established maximum pruning factor based on the probabilistic query processing method reaches the recall requirement, which is validated through similarity search on real-world data.

Pruning method	θ	Probabilistic query processing time	#Candidates	recall	# successful accesses	Ad-hoc query processing time	#Candidates	recall	# Total accesses	# successful accesses	# baseline candidates in
Horizontal	0.1	11.17	10031.3	100	24.6	3.4	3806.7	100	9.45	2	5105.3
	0.3	11.54	13257.0	100	33.93	4.4	5163.3	100	19.39	4	7360.4
	0.5	13.39	14452.2	100	33.59	7.4	7822.5	99.9	26.29	8	9059.5
Vertical	0.1	14.0	11252.9	100	125	1.03	1142.2	51.3	2	2	5105.3
	0.3	9.8	12208.7	100	167	1.67	1926.9	64.3	4	4	7360.4
	0.5	11.0	14001.9	100	112	4.08	3998.9	93.6	8	8	9059.5
Diagonal	0.1	10.38	10378.3	99.5	79.69	1.24	1309.1	37.3	2.63	2	5105.3
	0.3	11.06	11512.7	100	104.58	1.99	2098.4	47.3	4.94	4	7360.4
	0.5	11.32	13003.1	99.7	76.84	3.52	3938.9	61.0	9.61	8	9059.5

Table 5.5: Query processing in pruned sorted pairwise index on LiveJ based with ϕ^* for $\varrho = 99\%, k = 20$.

Pruning method	θ	Probabilistic query processing time	#Candidates	recall	# successful accesses	Ad-hoc query processing time	#Candidates	recall	# Total accesses	# successful accesses	# baseline candidates
Horizontal	0.1	0.027	2.37	100	5.2	0.017	2.05	99.8	30.33	3	2.327
	0.2	0.050	2.39	99.8	6.8	0.031	2.39	99.8	50.63	6.7	2.706
	0.3	0.030	2.65	99.8	8.9	0.040	2.65	99.7	70.82	6.9	3.068
Vertical	0.1	0.040	3.63	100	53	0.008	1.52	93.5	3	3	2.327
	0.2	0.050	4.00	100	68	0.013	1.77	95.0	5	5	2.706
	0.3	0.192	4.46	100	90	0.016	1.96	94.1	7	7	3.068
Diagonal	0.1	0.037	2.32	99.1	8.09	0.018	1.56	90.8	30.28	3	2.327
	0.2	0.047	2.61	99.9	10.6	0.035	1.87	94.5	48.70	5	2.706
	0.3	0.055	2.84	99.5	14.3	0.048	2.10	94.6	67.61	7	3.068

Table 5.6: Query processing in pruned unsorted pairwise index based on Yago with ϕ^* for $\varrho = 0.99$, $k = 20$.

- The ad-hoc query processing does not ensure the recall requirement, though retrieves less candidates than the probabilistic query processing, except for cases where the total scan almost matches with $E(l_Y)$.
- The diagonal pruning method always retrieve less candidates compare to other pruning methods, while using the maximum pruning factor established theoretically.
- The optimized pruning factor is higher when the distance threshold in similarity search is lower.

5.7 Summary

In this chapter, we discussed three different ways to prune inverted indices and presented a probabilistic analysis of the effect of index pruning to query processing, by exploiting the relatedness of inverted indices to locality sensitive hashing (LSH). Based on this analysis, we were able to formalize an optimization problem that determines the optimal pruning factor, considering a user-defined recall requirement and cost of query processing. Experimental evaluations were conducted over two case studies that validate our proposed query approach. The evaluations showed that the optimal pruning factor is quite high ($\geq 80\%$), when the distance threshold in similarity search is low ($\theta \leq 0.3$), and ad-hoc query processing in horizontal pruning outperforms for the optimal pruning factor in both scenarios.

Chapter 6

Entity-Centric Category Mining

6.1 Introduction

This chapter is based on our own publications at EDBT 2016 [PM16a] and SSDBM 2018 [PM18]. Here, we address the second main problem considered in this thesis; determining interesting attributes, in order to categorize a entity list accordingly, to understand and explore large and heterogeneous data *without prior domain knowledge*. Consider, for instance, knowledge bases, like YAGO, Freebase, or DBPedia, contain hundreds of millions of facts for millions of entities from all kinds of domains—leave alone information on the web in general. One classical approach to make such vast amounts of data easily accessible to users is to organize items into specific categories according to their attributes, in order to constrain the view of data explorers to such subsets. But who defines such data dimensions of interest?

Consider the case of large businesses with several retail stores across the USA. Clearly, data analysts are likely to investigate properties like the best selling items overall, but in particular also the best selling items per state or city. Likewise, it might also be interesting to investigate the sales of retail stores for specific product categories, or deals accomplished per employee. Analysts frequently use the drill-down operation in OLAP [GCB+97, SAM98] over predefined dimensions to analyze such cases. For a reasonably small scenario with well-defined schemata, telling which categories (dimensions) are interesting can be accomplished by domain experts, manually [JGP16]. However, when turning our attention to arbitrary, per se unknown scenarios in the age of Big Data, heterogeneity, dynamics, and scale strongly advocate solely automated means.

The overall task is the following: Given a table that contains data objects and their attributes, we want to determine those categorical attributes (i.e., columns of the table) that can be used to categorize the objects, thus, providing

Building	City	Country	Height
Burj Khalifa	Dubai	UAE	828m
Shanghai Tower	Shanghai	China	632m
Abraj Al-Bait Clock Tower	Mecca	Saudi Arabia	601m
Ping An Finance Centre	Shenzhen	China	599m
Goldin Finance 117	Tianjin	China	596m
One World Trade Center	NY City	United States	541m

Table 6.1: The World's Tallest Buildings (Wikipedia).

more focused and comprehensible information to users.

Applications scenarios are manifold, ranging from mining interesting patterns in data [WYY00, BKS10] or understanding dependencies between table columns [AIS93], to clustering or filtering entities or identifying dimensions of interest in data warehousing applications [DP13, DPD14, JGP16].

Let us introduce the problem through an example. Table 6.1 is showing part of a Wikipedia table reporting on the World's tallest buildings, sorted by height. This list is quite long, as *very many* tall buildings from many countries all over the World are captured. A refined view of this large table can be defined by imposing a constraint on the attribute *country*, such as country='United States' or country='China'. Browsing through such constrained tables fosters exploration/understanding of datasets at hand and can further answer specific information needs of users.

However, are all attributes useful in the sense that they define interesting subsets? With domain knowledge, it is a relatively simple task for humans to decide whether a categorical attribute is interesting for categorizing a list of entities, although sometimes subjective. For instance, categorizing skyscrapers by continent seems very reasonable, while categorizing them by architect depends on the number of skyscrapers per architect—boring if each architect designed one or two skyscrapers only. With large and heterogeneous data available, specifically on the web, hiring domain experts annotating attributes manually is infeasible. In this work, we propose a fully automated framework to learn a classification model that can identify suitable categorical attributes for categorizing entities. Suitable in the sense of human-perceived interestingness.

The human perception of 'interestingness' is a complex concept that asserts *generality, unexpectedness, conciseness, utility, and diversity* [GH06]. We first investigate existing probability-based objective measures of interestingness for categorical attributes [GH06, KP07]. However, we observe by anecdotal evidence and later also by experimental results that, in many cases, these measures fail to capture some aspects of data that are important in our task to identifying suitable attributes for categorization, discussed in Section 6.3.2. On the other hand, statistical measures such as variance or divergence, for finding interesting distribution of numeric data, are not directly applicable to our scenario. Moreover, as our objective is building a completely automated framework to train

the classifier, we leave out the discussion on utility-based measures that require user intervention.

Addressing the identified shortcomings in existing measures, we propose three novel statistical measures, p-diversity, p-peculiarity, and max-info-gap for categorical attributes, and finally, we learn a robust and accurate classification model using support vector machine (SVM) over combinations of proposed and existing measures. Based on a user study, we show that the trained classifier can predict those categorical attributes that are suitable for further categorization of entities—in a human-perceived sense. Experimental results also show that the proposed statistical measures are more effective in capturing 'interestingness' compared to existing measures like entropy or coverage. To the best of our knowledge, this is the first full-fledged approach that identifies meaningful categorical attributes, a generic and widely applicable component in data exploration and analytics.

6.1.1 Problem Statement and Notation

Our objective is to identify the categorical attributes that are perceived suitable by humans to define meaningful subsets of the entities for a specific entity-centric table. In this work, *we call such categorical attributes 'interesting'.*

To achieve this goal, we investigate a set of tables $\mathcal{R}$, where a table $r \in \mathcal{R}$ represents a list of entities and a set of associated attributes $\mathcal{A}$. We use a set of statistical measures $\mathcal{F}$ in order to map each categorical attribute (i.e., column of a table) to a feature space for training a classifier $\mathcal{C}$ that can predict if an attribute is interesting or not.

For example, consider the column country from Table 6.1 and denote the set of values for the attribute country with $\mathcal{V}_{\text{country}} = \{\text{UAE (1)}, \text{Saudi Arabia (1)}$ $\text{China (3)}, \text{United States (1)}\}$. The numbers in parentheses express the multiplicity. Now, we compute the statistical measures over $\mathcal{V}_{\text{country}}$, for instance, the Shannon entropy of the frequency distribution of $\mathcal{V}_{\text{country}}$ would be 1.792. If we knew that the attribute country is an interesting one for categorizing tall buildings, then we could, roughly speaking, learn that an entropy around 1.792 might be an indicator for interesting attributes, in any table we encounter.

To bring this toy example to larger scale, in order to build a reliable classifier, we face two main challenges:

- To the best of our knowledge, there is no catalogue available in literature, listing categorical attributes that are useful for the categorization of a specific entities class. Such training data is, however, crucial to train a classifier $\mathcal{C}$, thus, needs to be acquired first—and we want to do so without any human intervention.

- We have to identify suitable measures (statistical features) that can capture the characteristics of categorical attributes, tailored to our context of interestingness. Existing measures have certain drawbacks that hinder

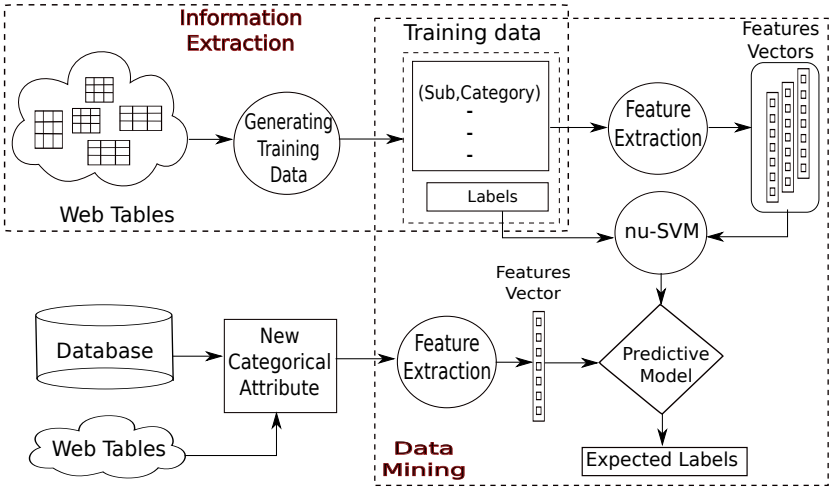


Figure 6.1: Framework of mining categorical attributes.

them grasping important characteristics.

Notation: An attribute $a \in \mathcal{A}$ of a table is divided into three types—(i) the subject of the table, denoted as a_s, which represents the class of the entities, (ii) the measuring attributes, denoted as $a_n \in \mathcal{N}$, which can be used for ranking criteria to order the subject entities, and (iii) the categorical attributes, referred as a_c. Additionally, for each $r \in \mathcal{R}$, we extract metadata M that holds the unique table identification, the constraint, and the ranking criterion of a table, denoted as r_{id}, r_{cons}, and r_{cr}, respectively, if available. Following this notation, the table r depicted in Table 6.1 is represented as $r = (a_s, a_n, a_{c1}, a_{c2}, M)$, where a_s = building, a_n = height, a_{c1} = city, and a_{c2} = country. No constraints are extracted for this table. Each attribute $a \in \mathcal{A}$ is associated with a value set, denoted $\mathcal{V}_a$. An example of $\mathcal{V}_a$ has been shown earlier in this section.

6.1.2 Sketch of our Approach

Our whole approach is divided into two main components, as shown in Figure 6.1. The *Information Extraction* component first creates the training samples from Wikipedia tables in a completely automated manner. Whether categorical attributes retrieved from a Wikipedia tables will considered as 'interesting' or 'not-interesting', for further categorization of the entities present in the table, is determined based on a central hypothesis, proposed in Section 6.2.

After labeling the categorical attributes, the *Data Mining* component extracts the feature vector $\mathcal{F}$ for each categorical attribute in the training data. $\mathcal{F}$ comprises the commonly used existing statistical measures like entropy, unalikeability, and newly proposed statistical measures, discussed in Section 6.3. Then, a classifier $\mathcal{C}$ is trained over the extracted feature vectors using ν-SVM. We evaluate how well these existing and proposed features can train the classi-

fier, individually or in combination, based on a user study.

6.1.3 Contribution and Organization

With this work, we make the following contributions:

- We present a framework to harness training samples of interesting and not-interesting categorical attributes from web tables without explicit human interaction.
- We propose three new statistical measures that can capture the interestingness of categorical attributes, tailored to our main objective and by reflecting the weaknesses of existing statistical measures.
- We have conducted a comprehensive evaluation, including a user study, demonstrating the applicability of the general approach and the superiority of the newly proposed features.
- The sample training data retrieved from Wikipedia tables, relevance assessments, and trained classifiers are made public and can be downloaded from `http://dbis.informatik.uni-kl.de/catmining/`.

The remaining chapter is organized as follows. Section 6.2 proposes the working hypothesis and algorithm to extract training data. Section 6.3 discusses the proposed novel statistical measures and introduces the learning model. Section 6.4 presents the experimental results. Finally, Section 6.5 summarizes the work.

6.2 Automated Extraction of Training Data

Here, we describe a fully automated way of collecting the training data from a set of tables. The training data hold a list of categorical attributes and the corresponding labels (interesting or not-interesting) that indicate whether the attribute allows a suitable categorization of the entities or not, reflecting a human notion of meaningful categorization. Hence, we need to determine the label for a categorical attribute a_c (i.e., a column of a table) that appears in a table for entity type a_s, but how can we determine the label without any human effort?

We put forward the working assumption that the presence and absence of web tables is an indicator of general interest or disinterest of humans in such tables. Following this assumption, the presence of a web table makes a categorical attribute interesting if that attribute is used as a constraint to create that very table. We cast this observation into our general hypothesis, given below. We will see later by experiments using human relevance assessments that this hypothesis is in fact well-grounded.

Building	City	Height
One World Trade Center	New York City	541m
Willis Tower	Chicago	442m
432 Park Avenue	New York	426m
Trump Tower	Chicago	423m
Empire State Building	New York City	381m

Table 6.2: List of Tallest Buildings in the Unites States (Wikipedia)

Hypothesis 1 *A categorical attribute in a table is considered interesting, thus, its statistical features are positive training samples, iff we find at least one table over the same entity class that is created by imposing a constraint over that categorical attribute.*

Let us walk through an **example** to explain the intuition behind this hypothesis. In Table 6.1, we observe that entities of class 'building' are displayed, together with the categorical attributes 'country' and 'city'. By browsing through Wikipedia, we also find another table, the *List of Tallest Buildings in the United States*, shown in Table 6.2. Clearly, both tables are created on the same entity class building and the constraint 'country=United States' is used in Table 6.2. This implies that we found (at least) one table (i.e., Table 6.2) which is created by imposing the constraint 'United States' on the categorical attribute 'country' from Table 6.1. Hence, according to our hypothesis, 'country' column of Table 6.1 is labeled 'interesting' for entity class 'building'. Here, we consider Table 6.1 as the parent table and Table 6.2 as the child table. Note that the child table (cf., Table 6.2) may not be a subset of the parent table (cf., Table 6.1), this is irrelevant for our task, however.

Once we have found such a parent-child pair of tables, we extract statistical features, for instance, information entropy, from the frequency distribution of the categorical attribute 'country' of the parent table and consider it a positive sample in our training data.

Although the final classifier is independent of specific entity types, while generating the training data from tables, an association between subject a_s and categorical attribute a_c is required. Because a categorical attribute can be associated with many entity types (e.g., 'length' can be an attribute for highways, bridges or beaches), thus, the pair (a_s, a_c) provides a unique identification for features retrieved for attribute a_c for entity class a_s. It would be misleading, or simply wrong, to search for *any* table (irrespective of matching entity type) that was generated by using a constraint on categorical attribute a_c, to conclude anything useful from the statistics computed from any table that has such an attribute a_c. *The final classifier is independent of the entity class* as it operates solely on statistical measures retrieved from categorical attribute.

6.2.1 Training Samples Generation Algorithm

To retrieve the label for categorical attributes and its statistics, a brute-force method would visit all pairs of tables to find the existence of a parent-child table pair. Avoiding the brute-force method, Algorithm 4 scans the tables twice to retrieve the training samples.

In the first table scan, the metadata such as constraint r_{cons} and subject a_s for all tables are extracted and a simple index, called *cons_map*, is created. The *cons_map* uses the constraint r_{cons} retrieved from a table as key and corresponding subject a_s of that table as value. For example, consider the two tables *"List of Tallest Buildings in the United States"* and *"List of Universities in the United States"* that are created using the constraint *United States*. Hence, the corresponding entries in *cons_map* is stored as:

$$\text{United States} \rightarrow \{\text{buildings, universities}\}.$$

The subject and the constraint of a table are extracted from the title of the table—more details on the extraction of metadata are discussed in Section 6.2.2.

After creation of *cons_map*, the second scan over the tables is done to retrieve all the attributes and their statistics. During this scan, we ignore numeric attributes (cf., Line 8 in Algorithm 4) and retrieve only categorical attributes for a specific entity class and store them as pair of (a_s, a_c) with its values V_{a_c}. Using these retrieved information together with *cons_map*, we generate the label for each (a_s, a_c) pair based on Hypothesis 1 and collect the statistics from V_{a_c} to create the training sample.

According to the proposed hypothesis, the label of a categorical attribute is found by identifying the existence of a parent and a child table that are linked by the categorical attribute. To determine the link, we scan each value in V_{a_c} associated with a categorical attribute a_c for a specific entity class a_s in a table and check the *cons_map* until we find a match. Once a value from V_{a_c} is found as a key in *cons_map*, we scan the subject list associated with the key to see whether the entity class a_s exists in the list. If the list contains a_s, we know that there exists another table which is built over the same entity class a_s by using one of the categorical value from V_{a_c} as constraint. This indicates that there exists a child table using a_c and we identify the current table that we are scanning as the parent table. Hence, according to our hypothesis, we label the pair (a_s, a_c) as 'interesting' and consider it as a positive training samples. If none of the values from V_{a_c} matches with the key in *cons_map*, we know that there are no child tables exist for the attribute a_c. Hence, the pair (a_s, a_c) is labeled as 'not-interesting' and is considered as a negative training sample.

While retrieving V_{a_c} for a categorical attribute a_c, we also capture the number of entities associated with each categorical value in $\mathcal{V}_{a_c}$ (e.g., in Table 6.2, $\mathcal{V}_{city} = \{\text{New York City (2)}, \text{Chicago (2)}\}$). This information is then used to map a pair (a_s, a_c) to feature space $\mathcal{F}$, capturing the empirical characteristics, such as information entropy, of the pair (a_s, a_c); discussed in detail in Section 6.3.

Note that web tables are not always fully consistent in data representation

Algorithm 4: Generating Training Samples

Data: Initialization
$cons\_map : \{key : constraints, value : subjectList[]\}$
training samples: $interesting[]$ `// a list of` $\{(a_s, a_c), \mathcal{F}\}$ `where` a_c `is used to categorize` a_s`.`
Negative training samples: $notInteresting[]$ `// a list of` $\{(a_s, a_c), \mathcal{F}\}$ `where` a_c `is not used to categorize` a_s

Procedure `generateSamples(`*web tables* $\mathcal{R}$`)`
  `/* Scan on` $\mathcal{R}$ `to build` *cons_map* `*/`
  for $r \in \mathcal{R}$ **do**
    $r.a_s, r_{cons} \leftarrow parse\_metadata(r)$
    add $(r.a_s)$ to the list of $cons\_map[r_{cons}]$
  end for
  `/* Scanning` $\mathcal{R}$ `to build training samples */`
  for $r \in \mathcal{R}$ **do**
    $List\{r.A\} \leftarrow parse(r)$ `// Parsing all colums`
    $List\{r.A_c\} \leftarrow List\{r.A\} \setminus (r.a_s \cup \mathcal{N})$ `// Removing numeric attributes`
    for $a_c \in List\{r.A_c\}$ **do**
      $\mathcal{F} \leftarrow calculateFeatures(\mathcal{V}_{a_c})$
      `/* Find existence of parent, child table in` $\mathcal{R}$ `based on` $\mathcal{V}_{a_c}$ `*/`
      for $x \in \mathcal{V}_{a_c}$ **do**

        $subjectList \leftarrow cons\_map[x]$
        if $r.a_s \in subjectList$ **then**
          add $\{(r.a_s, a_c), \mathcal{F}\}$ to $interesting[]$
          break
        else
          add $\{(r.a_s, a_c), \mathcal{F}\}$ to $notInteresting[]$
        end if
      end for
    end for
  end for
  return $interesting[], notInteresting[]$

and column descriptions [SC14, IRW16]. As ambiguous representations of numeric types can generate wrong classification labels to categorical attributes, Algorithm 4 excludes all attributes of numeric type. Such restriction removes categorical attributes such as years, while generating only the training data, but not from the test data. The trained classification model is applied to any categorical attribute, independently of its type.

6.2.2 Harnessing Wikipedia

In this work, we specifically use the English Wikipedia corpus to generate the training samples. The major difficulties in extracting and understanding information from web tables arise due to inherent heterogeneity in schema and data representation [CP11, WWWZ12].

Here, we discuss in more detail why Wikipedia is an excellent source for our endeavor. First of all, by enforcing collaborative editing policies and controlling duplicated information across multiple pages, Wikipedia maintains high information quality, thus, *is generally considered credible for knowledge exploration* [CLK$^+$16, MW08, BND13]. For this work, we also excluded user pages to avoid biased data.

Second, Wikipedia tables contain surprisingly many categorical attributes. Such information is not available in web portals, like `http://rankopedia.com` or `http://ranker.com`, where tables are created based on crowdsourcing, with only numeric attributes (mainly number of upvotes) being available for the entities. More precisely, 3/4 of all Wikipedia tables that are investigated for this work contain categorical attributes. The distribution of the number of categorical attributes per table can be well described by a Poisson distribution with mean $\lambda = 1.9$ (with relative sum squared error of < 0.00001).

Third, the structure of the tables in Wikipedia is quite consistent and the metadata of tables, required by Algorithm 4, can be extracted. However, we noticed that often Wikipedia tables do not have sufficient information about table description associated via html tags, not even the title of the table. Hence, retrieving metadata from arbitrary tables would require sophisticated NLP techniques and further human involvement for checking the correctness of the retrieved results. To avoid the human involvement, in this work, we consider only those tables that have the property of being sortable and are extracted from the pages that has page title beginning with the phrase "List of ...". This greatly helps to accurately collect metadata, such as subject and constraint of the table, by parsing the title/caption of the table or the title of the Wikipage. These page titles have a very simple sentence structure that can be easily parsed by using *propositions from the English dictionary* to retrieve the subject and the constraints of a table. For example, from the page title "List of Tallest Buildings in the Unites States", we retrieve subject of the table as 'Tallest Buildings' and 'Unites States' as the constraint, using prepositions 'of' and 'in'. Although sometimes page titles are more complex than the given example, they are still easily parseable and much less complicated than full-fledged sentences in regular text paragraphs.

Now, we will discuss briefly how we identify the subject and the categorical attributes/constraint from a table. We use the subject parsed from the page title discussed earlier as a hint to identify the subject column of the table. To do so, we check whether any of the column headers of the table matches with the subject retrieved from the page title. The match is considered true if any of the stemmed words (nouns) retrieved as subject from the page title matches

with the stemmed column headers. Then, the matched keyword is considered the subject for the training sample and the corresponding column is identifies as subject column for that table.

There are few cases where the subject obtained from the page title and the header of the subject column in the table do not use the same common noun for the subject. For example, in many cases, we found the table header of the subject column is referred as 'name'. In such cases, we use the retrieved subject from the page title as the subject for the training sample and the adjacent column to the sortable column (ranking column) in the table as the subject column. We filter the numeric attributes during our sample generation as presented in Line 8 of Algorithm 4. To do so, we employ a dictionary of units to recognize numeric attributes, i.e., if the table header or cells contain 'lbs' or 'kg'. The presence of only numeric content in a cell of a table column is also used to identify the numeric attribute. Although the structure of tables are well defined in Wikipedia, the data are not free from ambiguity. For instance, in some tables, numerical data (e.g., age) are spelled-out, thus, are retrieved as non-numeric categorical values. Such cases yield false identification of a numeric attribute as a categorical attribute in our training data. More sophisticated extraction methods [LSC10, BND13] could be used in Algorithm 4 for metadata extraction, which is orthogonal to our proposed hypothesis. However, despite this restriction in obtaining training data, the trained classification model can in fact predict interesting categorical attributes of numeric type, such as years.

Potential Limitations: Not surprisingly, in various cases, the absence of a table in Wikipedia happens due to the limited manpower and not due to general disinterest in that table. In fact, we found cases where a table is missing in Wikipedia where human evaluators in our user study unanimously state that they are interesting and, following our hypothesis, should exist. For instance, the list of the gold medalists in Olympic history, grouped by the type of sport, was not present in Wikipedia at the time of harnessing the training data, but marked interesting by the majority of voters in our user study. In fact, during the progress of this work, we found that several such lists of gold medalists were added to Wikipedia. Due to this characteristics, the accuracy of the samples extracted by Algorithm 4 suffers from false positive data and reaches only 68.9% overall accuracy according to the user assessments. However, even though few training samples were apparently misleading, our classifier is able to correctly classify the task according to the evaluators' judgments. Supported by such exemplary evidence, and the overall performance shown in the experimental evaluation, we believe the hypothesis is reasonable to generate the training data for our learning task, *as important tables are created supposedly before people spend effort in creating less important ones.*

We will see later that by directly considering the output of the described extraction algorithm, i.e., the training samples, we obtain a 68.9% accordance to human assessments. Whereas, the performance of our classification model, learned based on generated training data, reaches 90.9% classification accuracy.

Algorithm 5: Map-1(a)

```
1  Procedure map1(key1, value1)
2      List{r.A} ← parse(r)
3      List{r.A_c} ← List{r.A} \ (r.a_s ∪ N)
4      for a_c ∈ List{r.A_c} do
5          for x ∈ V_{a_c} do
6              key2 ← (x, r.a_s)
7              value2 ← (r_id, a_c, count)
8              return emit(key2, value2)_cat
9          end for
10     end for
```

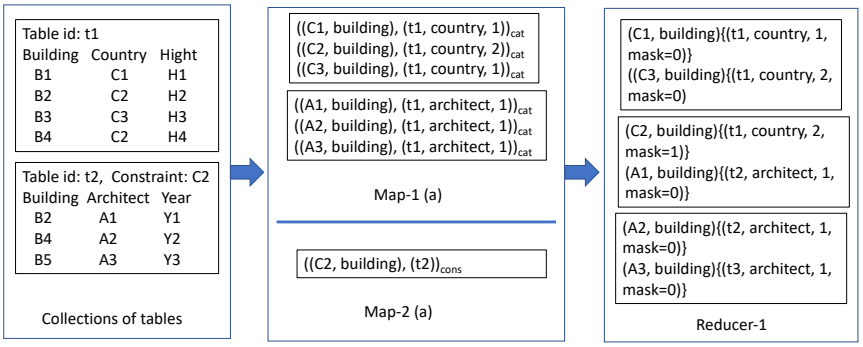


Figure 6.2: The first phase of MapReduce job with a toy example.

6.2.3 Scale-Out Adaptation of the Generation of Training Data

In this section, we present a scale-out adaptation of the generation of training data, using the MapReduce framework [DG04], in order to make the Algorithm 4 efficient, while dealing with large web corpora. MapReduce is a simple programming model that allows processing data distributed across clusters. The MapReduce implementation of Algorithm 4 requires two sequential MapReduce jobs. Normally, each table is small enough so that we can avoid distributing it over clusters. Thus, we can reduce the communication overhead of counting appearance of a particular categorical value in a table, denoted as *count* in Line 7 of Algorithm 5, an important parameter for the function $calculateFeatures()$ in Algorithm 4. Algorithm 5 processes each ranking and emits a map that keep the count for a categorical value appearing in a specific table. Additionally, we associate tag_{cat} for each emitted value to create a lineage from this map job. After computation of the first map job (Algorithm 5), the combiner produce the list of tables under the same key.

We use another map job to create *cons_map*, mentioned in Algorithm 4. Us-

ing Algorithm 5 and Algorithm 6, we basically create two different lineages tag_{cat} and tag_{cons}. After that, Algorithm 7 computes the left outer join of these two lineages, which can be implemented following the discussion in paper [YDHJ07]. It allows us to find if a categorical value from a table (in $key2$) appears as a constraint in another table or not. We keep track of this information by introducing a Boolean variable *mask* for each categorical value. Figure 6.2 presents an illustration of the first phase of MapReduce job on a simple example using two small tables. Using Algorithm 5 and Algorithm 6, statistics for each categorical value appearing in two example tables and *cons_map* are created. These results are then joined by Reducer-1 (Algorithm 7) to produce intermediate results for the next phase of MapReduce job, shown in Figure 6.2.

Algorithm 6: Map-1(b)

```
1  Procedure map2(key1, value1)
2      r.a_s, r_cons ← parse_cons(r.M, r)
3      key2 ← (r_cons, r.a_s)
4      value3 ← (t_id)
5      return emit(key2, value3)_cons
```

Algorithm 7: Reducer-1: Generating intermediate results

```
1  Procedure reducer(key2, list < values >)
2      for v ∈ list < values > do
3          if v.tag = tag_cons then
4              flag ← true
5              drop v from list < values >
6          end if
7      end for
8      if flag = true then
9          mask ← 1
10         output ← (list < values >, mask)
11         return emit(key2, output)
12     else
13         mask ← 0
14         output ← (list < values >, mask)
15         return emit(key2, output)
16     end if
```

According to our proposed hypothesis, if any categorical value for a specific entity class (i.e., any categorical value from a specific table) appears at least once as a constraint in another table, the corresponding categorical attribute is considered interesting. This condition is checked easily by a set containment query in Line 13 of Algorithm 4. But, in case of the scale-out adaptation of Algorithm 4, we need to pay attention here that the output from Reducer-1

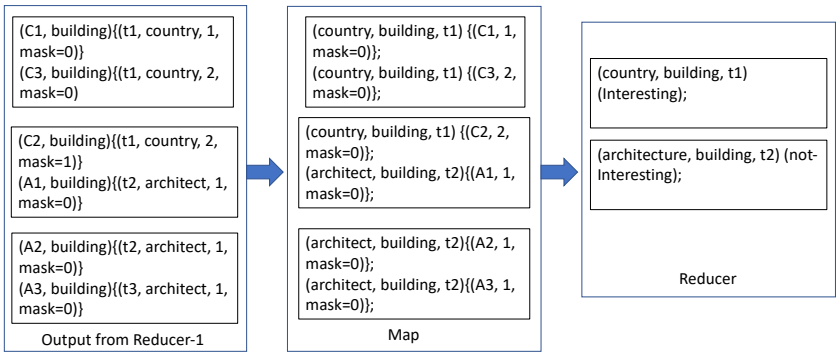


Figure 6.3: Exemplary illustration of Algorithm 8.

(Algorithm 7), though seems similar like the the set containment query, can not actually produce the final results. As the output of Reducer-1 is computed based on the keys ($key2$), partitioned by categorical values, the MapReduce framework can not ensure that all categorical values from a specific table will appear in the same partition. Hence, according to our hypothesis, we cannot decide whether *at least one* categorical value from a table is used as constraint in another table, based on the output generated by Reducer-1, independently from each partition. To deal with this problem, we require another MapReduce job, presented in Algorithm 8, in order to generate the final results, i.e., the label ('interesting' or 'not-interesting') for categorical attributes in the training data.

Algorithm 8: Generating label for retrieved categorical attributes.

Procedure `map`($key2, list < values >$)
  for $v \in list < values >$ **do**
    $key3 \leftarrow (v.a_c, key2.a_s, v.r_{id})$ $value \leftarrow (key2.x, v.mask)$
    return $Emit(key3, value)$
  end for
Procedure `reducer`($key3, list < values >$)
  for $v \in List < values >$ **do**
    $sum = sum + v.mask$
  end for
  $key_{final} \leftarrow (key3.a_c, key3.a_s)$
  if $sum > 0$ **then**
    return $emit(key_{final}, interesting)$
  else
    return $emit(key_{final}, notInteresting)$
  end if

The `map` procedure in Algorithm 8 simply re-arrange the information of

categorical values, generated by Reducer-1, where the table id (r_{id}) associated with a categorical value is considered as $key3$. This ensures that the results of set containment query for all categorical values from a table end up in the same partition. Then, the reducer in Algorithm 8 generates the label 'interesting' for a categorical attribute if any of the categorical values in a specific table holds $mask = 1$, which signifies that the categorical value is used as a constraint in another table. Using the output from the Reducer-1 (cf., Figure 6.2) for the toy example, Figure 6.3 presents the second phase of MapReduce job based on Algorithm 8.

6.3 Learning the Classification Model

After creating the training samples based on the proposed hypothesis, we step toward to learn the classification model. For the task of learning which categorical attributes are interesting, we use different statistical measures to capture essential characteristics of the frequency distribution of values inside table columns. As discussed earlier in Section 6.1, *interestingness is not a fixed concept.* It is rather a meta concept capturing various separate concepts like *generality*, *coverage*, *reliability*, *peculiarity*, *diversity*, and *utility* of data. Here, we propose three statistical measures, coined *p-diversity*, *p-peculiarity*, and *max-info-gap*, capturing drawbacks of four existing statistical measures, *entropy*, *max-coverage*, *unalikeability*, and *peculiarity*, that are commonly used to capture different characteristics of categorical attributes. The features are first extracted from the training samples using these measures and then the ν-SVM approach is used to train the classifier.

Before we dive into the concrete definitions of the individual measures, we present Table 6.3, in order to understand better on which data these measures are actually executed. In the heading of the table, we values and their frequencies (i.e., the set V_{a_c}) for three made-up exemplary tables for a categorical attribute 'country'. We see that the different measures (normalized in [0,1]) vary quite strongly, relative to each other, but also compared to the same measure for different table characteristics. For instance, the normalized entropy ($\hat{H}$) of Example 2 is 0.79, signifying average information content in Example 2 is high, as the frequency of each country is uniformly distributed. Whereas the entropy of Example 1 is 0.44, signifying that the average information content in Example 1 is lower than Example 2 due to the skewness in categorical values. On the other hand, the max-coverage value of Example 2 is 0.18, very different from its entropy value, quantifying the maximum dominance of a categorical value within it (quite low for this example). The key point is that individual measures highlight different aspects of categorical values—for instance, the degree of randomness or dominance.

	Example1 USA(12) Spain(8) Germany(2) China(2) Australia(2) France(2)	**Example2** USA(2) Germany(2) China(2) Australia(1) France(2) Switzerland(1) Russia(1)	**Example3** USA(12) Germany(2) China(2) Australia(2) France(2) Switzerland(1) Russia(1)
$\hat{H}$	0.44	0.79	0.48
$mCov$	0.43	0.18	0.55
mIg	0.75	0.29	0.8
U	0.71	0.84	0.67
D	0.74	0.93	0.7
$p\hat{P}ec$	0.53	0.46	0.64
$p\hat{V}ar$	0.36	0.7	0.49

Table 6.3: Sample Data and Corresponding Measures.

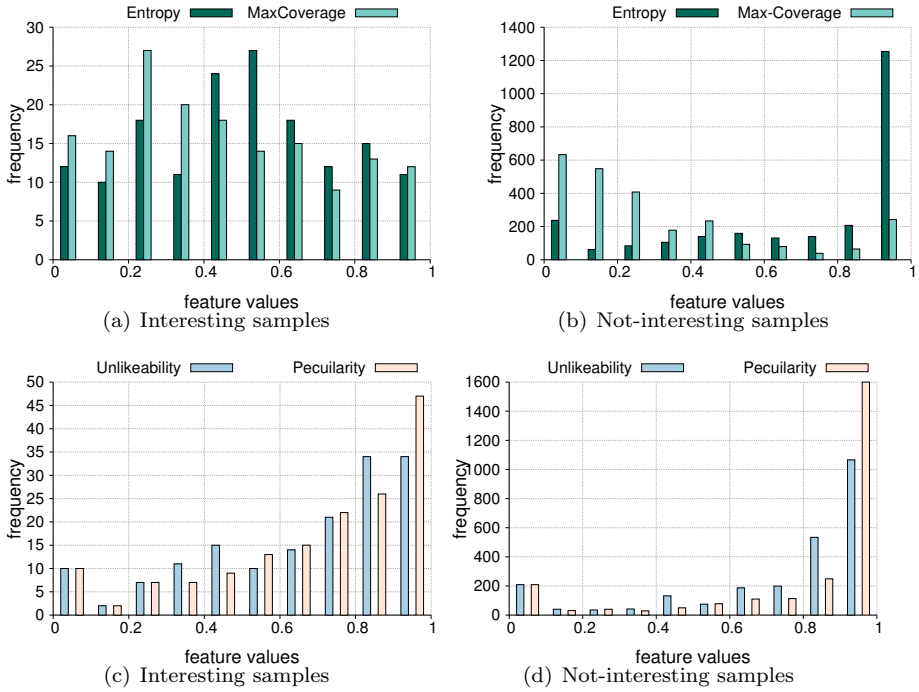
6.3.1 Existing Features for Categorical Attributes

Several probability-based objective measures for interestingness are proposed in literature, specifically for mining association or classification rules, capturing the generality or reliability of such rules. One of the most prevalent measures is *entropy*, mainly used for mining attribute-value pairs in decision trees. Statistical measures that are capturing diversity of categorical attributes are, on the other hand, less prominently investigated [DS08]. Below, we briefly discuss how the traditional statistical measures, discussed in Section 2.2.2, can be used as features to learn the classification model and how these measures can reflect the interestingness of a categorical attribute.

Shannon Entropy: In this work, a categorical attribute a_c is treated as a random variable where $\mathcal{V}_{a_c}$ represents the set of possible values that a_c can hold. Shannon entropy for a_c is calculated by $H(a_c) = -\sum_{x \in \mathcal{V}_{a_c}} P(x) \log_2 P(x)$, with $P(x) = count(x)/|\mathcal{T}|$, where $|\mathcal{T}|$ is the size of the table and $count(x)$ is the frequency of value $x \in \mathcal{V}_{a_c}$. We use the normalized entropy, given by:

$$\hat{H}(a_c) = \frac{-\sum_{x \in \mathcal{V}_{a_c}} P(x) \log_2 P(x)}{\log_2 |\mathcal{T}|}.$$

Here, $\hat{H}(a_c) = 1$ when each categorical values appears only once, and $\hat{H}(a_c) = 0$ when a_c holds only one value for all entities. Clearly, a piece of information is considered interesting when the randomness of the information content is neither extremely high (i.e., $\hat{H}(a_c) = 1$) nor extremely low (i.e., $\hat{H}(a_c) = 0$). In Figure 6.4(a) and Figure 6.4(b), we present the distribution of the entropy values extracted from retrieved interesting and not-interesting samples respectively. Following the interpretation of entropy in our context,



(a) Interesting samples (b) Not-interesting samples

(c) Interesting samples (d) Not-interesting samples

Figure 6.4: Distribution of features value

we found that 69% of the seemingly interesting samples (categories) have an entropy value in $[0.2, 0.8]$ (cf., Figure 6.4(a)), and 71% of the not-interesting categories have an entropy value in $[0, 0.2]$ or $[0.8, 1]$ (cf., Figure 6.4(b)).

Max-Coverage: According to the discussion in Section 2.2.2, max-coverage of a categorical attribute a_c is calculated as:

$$mCov(a_c) = max_{x \in \mathcal{V}_{a_c}}\{P(x)\}.$$

It captures dominance of a categorical value. If all the entities in the table have the same value for a_c, then $mCov(a_c) = 1$. Such an extreme case is definitely not an interesting one. On the other hand, $mCov(a_c) \rightarrow 0$ when very many categorical values are associated with a_c and each entity holds a different categorical value. This scenario is also not an interesting one. Intuitively, the mid range in [0,1] might represent $mCov$-value for a interesting category. According to the distribution of $mCov(a_c)$ values in our positive and negative training samples, we observe that Max-Coverage values lie in $[0.2, 0.8]$ for 66% of interesting category samples (cf., Figure 6.4(a)) and 47% of the not-interesting categories have a $mCov$-value in $[0, 0.2]$ (cf., 6.4(b)).

It should be mentioned here that a categorical attribute with a large entity list is suitable for the categorization of entities, when the attribute has a skewed frequency distribution with high $mCov$-value or a uniform distribution with low $mCov$-value. According to the interpretation of $mCov$-value, discussed

earlier, *mCov*-value for both mentioned scenarios fail to identify the categorical attribute as interesting for categorization.

Unalikeability: For capturing the *degree of diversity* of categorical attributes, we use the unalikeability measure, discussed in Section 2.2.2. Like entropy measure, considering a categorical attribute a_c as random variable, the unalikeability measure $U(a_c)$ is calculated as:

$$U(a_c) = 1 - \sum_{x \in \mathcal{V}_{a_c}} P(x)^2.$$

$U(a_c) = 0$ when a categorical attribute a_c holds the same value for all entities, thus, can not further refine the entity list. On the other hand, $U(a_c) \to 1$ signifies that a_c is too diverse to choose an attribute value pair for further refinement of entity list. Clearly, both cases represent that the category is not interesting in our context. Reflecting this characteristic, Figure 6.4(d) shows that 83% of not-interesting categories hold unalikeability values in $[0, 0.2]$ or $[0.8, 1]$.

This measure quantifies how often observations of a random variable differ from one another. Hence, two categorical attributes, having a similar frequency distribution but different population sizes, can not be distinguishable by this measure. For instance, uniformly distributed categorical values in a large table gets a unalikeability value near '1', and thus, is identified as not-interesting, which is definitely not the case. This characteristic is also reflected in the distribution of unalikeability values of retrieved training samples. We can see that unalikeability value of 46% of interesting samples lie in [0.8, 1.0] (cf., Figure 6.4(c)), which is making the distribution similar to the value distribution of not-interesting samples (cf., Figure 6.4(d)).

Peculiarity: Peculiarity is another diversity measure used in this work. It is the complement of the Simpson's index, commonly used in mathematical ecology, discussed in Section 2.2.2. Considering categorical attribute as random variable, peculiarity of a categorical attribute a_c, denoted as $D(a_c)$, is calculated as:

$$D(a_c) = 1 - \sum_{x \in \mathcal{V}_{a_c}} \frac{count(x)(count(x) - 1)}{|\mathcal{T}|(|\mathcal{T}| - 1)}.$$

$D(a_c)$ also shows almost a similar characteristic as $U(a_c)$, in the context of understanding the interestingness of a categorical attribute. From Figure 6.4(d) and Figure 6.4(c), we can see that the distribution of $D(a_c)$-values for both, not-interesting and interesting samples, respectively, have a similar distribution pattern with $U(a_c)$. Supporting our interpretation of $D(a_c)$-values in the context of identifying interesting category, around 74% of not-interesting categories hold feature values in $[0, 0.2]$ or $[0.8, 1]$ (cf., Figure 6.4(d)). This measure also fails to identify the scenario that a uniform distribution with large population would be interesting for categorizing an entity list.

6.3.2 Novel Features for Categorical Attributes

The existing measures discussed above are commonly used as impurity measures in classification methods, such as decision trees. Though they reflect information about the distribution of categorical values, they fail to capture some distinctive features of the distribution that can provide important insights for categorization of the entities. The problem we are addressing in this work calls for a more fine-tuned understanding of the distribution of categorical data. In the following, we propose three probability-based statistical measures: (i) max-info-gap, (ii) p-diversity, and (iii) p-peculiarity. We also discuss how these measures provide more distinctive information to understand which distribution of categorical values would be interesting in our context, compared to the above existing statistical measures.

Max-Info-Gap: Consider a specific value x, say 'China', of a categorical attribute a_c in a table. If x is very frequent, the information contained in that column of the categorical value is low. In the extreme case, a_c can hold one unique value for all entities. This could mean that the table is explicitly created for entities that hold this one specific categorical value. In information theory, the maximum amount of information content that a specific categorical value can hold is $-\log_2 \frac{1}{|\mathcal{T}|}$ for a table $\mathcal{T}$ with $|\mathcal{T}|$ rows; basically when it is describing only one entity in the table. Now, the idea behind max-info-gap is to quantify the maximum difference between the information content expressed by one specific categorical value within V_{a_c} and the maximum information content a categorical value can hold hypothetically for that table, given by:

$$max_{x \in V_{a_c}} \left\{ (-\log_2 |\mathcal{T}|^{-1}) - (-\log_2 P(x)) \right\}.$$

The maximum difference occurs for the categorical value with maximum coverage. Unlike max-coverage, which only focus on the maximizing $P(x)$, max-info-gap considers the size of the table to find more insights about the coverage value. It signifies the dominance of a categorical value considering the table size. With the existing notion of max-coverage, we define max-info-gap as follows.

Definition 9 *Max-Info-Gap is the maximum information gap between the maximum information that a categorical value can hold for the categorical attribute and the actual information it is holding. It is denoted as $mIg(a_c)$ for categorical attribute a_c, and is calculated as follows:*

$$mIg(a_c) = 1 - \frac{\log_2 mCov(a_c)}{\log_2 |\mathcal{T}|^{-1}}.$$

The values of $mIg(a_c)$ fall by definition into $[0, 1]$. When an attribute a_c is completely diverse, i.e., each value in a_c is exactly associated with one entity, $mIg(a_c) = 0$, clearly not an interesting scenario for further categorizing entities. For a fixed table size, as the dominance of one specific categorical value increases, $mIg(a_c)$ also increases. In general, we can say that a skewed distribution of

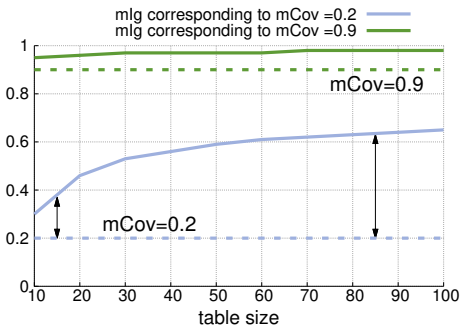


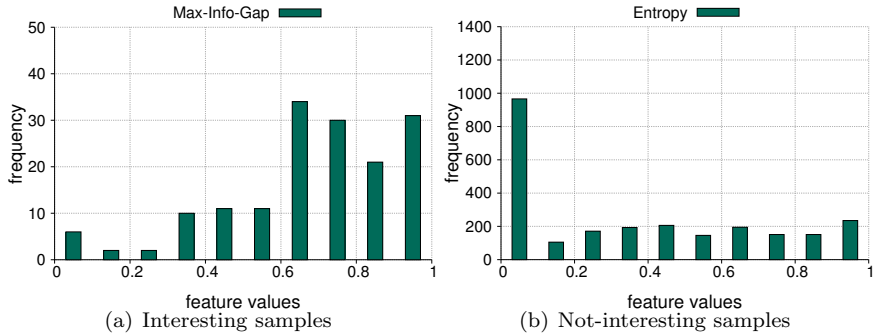
Figure 6.5: Comparison between max-info-gap and max-coverage with varying table size.

categorical values hold higher mIg-measure than the uniform distribution for a fixed table size. In extreme case, when all entities hold the same categorical value then $mIg(a_c) = 1$ as $mCov(a_c) = 1$. For example, if we compare the values of mIg-value in Table 6.3, we can observe that the mIg-value of Example 1 and Example 3 are higher than the mIg-value of Example 2 as the distribution of categorical values in Example 1 and Example 3 are more skewed than Example 2. We can also see that the mIg-value of Example 3 is slightly higher than the mIg-value of Example 1, as the dominance of 'USA' is higher in Example 3, considering that the table length is comparable in both examples .

Example 4					
USA	Spain	Germany	China	Australia	France
(60)	(50)	(45)	(60)	(40)	(60)

It is important to discuss here how significantly max-info-gap differs from max-coverage, as both of these measures quantify the dominance of a categorical value in a table. Both measures, mIg and $mCov$ hold a value towards 1 if one categorical value is very dominant. But mIg considers the table length to reward the $mCov$-value as table length increases. In Figure 6.5, we present how table length affects mIg and $mCov$ measure. We can see from the Figure 6.5 that for a higher coverage value, such as $mCov = 0.9$, mIg-values do not differ much from the $mCov$-values for different table sizes. This characteristic signifies the existence of one very dominating categorical value. On the other hand, for a low coverage value, representing a non-skewed distribution of categorical values, mIg-values significantly differ from $mCov$-value as the table length increases. In Figure 6.5, we can see that with a fixed low $mCov = 0.2$; for a small table with 10 entities holds $mIg = 0.3$, whereas a larger table with 100 entities $mIg = 0.63$, significantly higher to indicate that the further categorization of entities is suitable here. The $mCov$ measure cannot capture this insight from the distribution of the categorical values.

When comparing Example 2 and Example 4, we can see that Example 4 is intuitively more suitable for further categorization. Both examples have $mCov = 0.16$ but mIg-values of them differ significantly. For Example 4,



(a) Interesting samples (b) Not-interesting samples

Figure 6.6: Distribution of max-info-gap in retrieved training samples.

$mIg = 0.74$ which is high enough to emphasize the possibility of meaningful categorization, whereas $mIg = 0.28$ for Example 2 (cf., Table 6.3).

Figure 6.6 presents the distribution of mIg-values for retrieved training samples. From Figure 6.6(a), we observe that 39% of the not-interesting samples have mIg-value in [0, 0.1], reflecting that $mCov \rightarrow 0$ can be an indicator of not-interesting category, mentioned earlier. Moreover, 19% of the interesting samples have $mCov \leq 0.1$ in our training data (cf., Figure 6.4(a)), whereas only 3% of the interesting samples have $mIg \leq 0.1$ (cf., Figure 6.6). This observation reflects that max-info-gap can better distinguish interesting categorical attributes from not-interesting ones.

P-Diversity: To understand the deviation of the distribution of categorical values from a predefined reference distribution, we propose the p-diversity measure. In contrast to the concept of unalikeability, it describes how often the observation of a random variable varies with respect to an pre-established reference frequency. This reference frequency is defined based on how we describe the distribution of categorical values for meaningful categorizations. For a categorical attribute a_c, if all entities hold the same categorical value, there is no diversity among the observations (in such case, $U(a_c) = 0$). In this scenario, imposing a constraint on a_c cannot create a new and refined table, thus, is clearly not an interesting scenario for categorizing the entity list by a_c. First, the categorical attribute must hold at least two values for having a possibility of creating refined tables by putting a constraint over values of a_c. Second, in an ideal case, these two categorical values will be equally distributed over the entities in the table. Based on these two observations, we consider the reference frequency 0.5, in this work. It represents the minimum diversity for a categorical attribute to become interesting. Relative to this reference frequency, we define the measure p-diversity of a categorical attribute as follows.

Definition 10 *P-Diversity is the square root of the sum of squares of the differences between the actual coverage of a categorical value to the reference coverage*

value 0.5. *It is denoted as* $pVar(a_c)$ *for* a_c, *and is calculated as:*

$$pVar(a_c) = \sqrt{\sum_{x \in \mathcal{V}_{a_c}} (P(x) - 0.5)^2}.$$

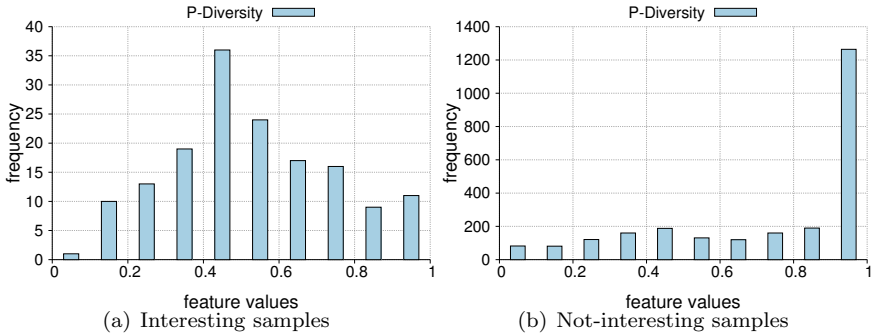
Note that the maximum $pVar(a_c)$ is equal to $(1 - 0.5|\mathcal{T}|)/\sqrt{n}$, which occurs when the categorical attribute holds different values for each entity in the table. The normalized P-Diversity is, thus, given by:

$$\hat{pVar}(a_c) = \frac{\sqrt{\sum_{x \in \mathcal{V}_{a_c}} (P(x) - 0.5)^2)}}{(1 - 0.5|\mathcal{T}|)/\sqrt{n}}.$$

Here, $\hat{pVar}(a_c) = 1$ if the categorical attribute holds different values for each entity in a table and $\hat{pVar} \to 1$ represent the cases where further categorizations are not suitable. On the other hand, a $\hat{pVar} \to 0$, while the coverage of categorical values tends to 0.5, which indicates the possibility of further categorization. The normalizing factor for p-diversity considers the size of the table and rewards $pVar$-value as the table size increases. Hence, a uniform distribution with a coverage value far from 0.5 might hold a $\hat{pVar}$-value close to 0, if the table size is large. Rewarding the table with larger size, even though the categorical values have a lower coverage, is perfectly in line with the consideration for meaningful categorization of a table, in our context. For example, let us consider Example 2 and Example 4 where both lists have almost similar distribution of categorical values. Additionally, the coverage of each categorical value is less than 0.2 in both lists, which is significantly smaller than the reference coverage of 0.5. But due to the large table size, Example 4 holds $\hat{pVar} = 0.09$, close to 0, indicating further categorization of the table, while Example 2 holds $\hat{pVar} = 0.66$, indicating not a meaningful categorization scenario. Unalikeability cannot capture this characteristic and holds almost similar U-values, 0.85 and 0.83, respectively, for Example 2 and Example 4, indicating both scenarios are not suitable for the further categorization of entity list.

We present the $\hat{pVar}$-value of retrieved not-interesting samples in Figure 6.7(b), where we can observe that that 50% of retrieved not-interesting training samples hold $\hat{pVar}$-values in range of [0.9, 1] and 72% not-interesting training samples have $\hat{pVar}$-value ≥ 0.5. Reflecting the significance of the reference distribution, Figure 6.7(a) shows that the density of interesting samples is highest at $\hat{pVar} = 0.5$. In line with our interpretation of $\hat{pVar}$-value, we also observe that 80% of the retrieved interesting categorical attributes hold $\hat{pVar}$-values within $[0.2, 0.8]$.

According to the definition of $\hat{pVar}$, a uniform distribution of categorical values is considered to be more useful for further categorization than a highly skewed distribution of categorical values where $\hat{pVar} \to 1$. We can see in Table 6.3 that less skewed Example 1 has lower $\hat{pVar}$ value compared to Example 3.



(a) Interesting samples (b) Not-interesting samples

Figure 6.7: Distribution of p-diversity in retrieved training samples.

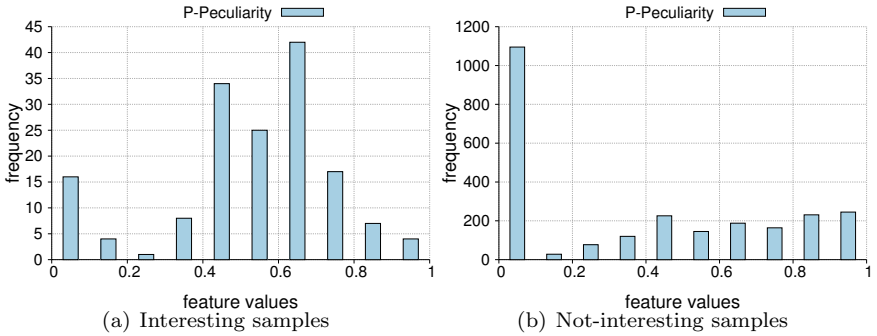
Table size	list	$p\hat{V}ar$	U	D
100	list1: { USA(80), Spain(20)}	0.09	0.32	0.32
	list2: {USA(60), Spain(40)}	0.03	0.48	0.48
5	list3: {USA(4), Spain(1)}	0.63	0.32	0.4
	list4: {USA(3), Spain(2)}	0.21	0.48	0.6

Table 6.4: Comparing $p\hat{V}ar$-value with unalikeability.

Now, we will discuss the difference between p-diversity and two existing diversity measures. Let us consider the example in Table 6.4. In the first row, we represent two lists where list1 is more skewed than the list2. The second row holds list3 and list4 with an identical distribution of values as list1 and list2, respectively, but both lists hold very few entities. As unalikeability does not consider the list size, it cannot distinguish between list1 and list3 as both lists have the similar distribution. In contrast to unalikeability measures, $p\hat{V}ar(list1)$ is very close to 0, indicating a possibility for further categorization of list1 and $p\hat{V}ar(list3) = 0.63$, which tells that list3 is not suitable for further categorization of its entities. On the other hand, the peculiarity measure increases as the table size decreases, consequently, a skewed distribution of categorical values in a small table is considered more interesting than a larger one. For example, Table 6.4 shows that $D(list3) = 0.4$ is closer to 0.5 than $D(list1) = 0.32$. As mentioned earlier, a D-value close to 0.5 characterizes interesting attributes, and thus, list3 is more suitable for categorization than list1 according to peculiarity, which is clearly not the case. In this table, we can also see that as list size increases, the difference between $p\hat{V}ar$-value for skewed and uniform distribution decreases as expected.

In the experimental study in Section 6.4, we will see that the classification model created by using p-diversity performs better than the model that is considering existing diversity measures.

P-Peculiarity: Finally, we propose the p-peculiarity measure to capture the unexpectedness in a categorical attribute of a table. A categorical attribute



(a) Interesting samples (b) Not-interesting samples

Figure 6.8: Distribution of p-peculiarity in retrieved training samples

does not show any peculiarity if the categorical values are uniformly distributed over the entities. With p-peculiarity, we measure the skewness of the data by finding the difference of its distribution from the uniform distribution. Unlike the max-info-gap, where the most skewed categorical value is used to quantify the skewness, here, *all* categorical values are considered, i.e., the actual frequency distribution of categorical values is considered.

Definition 11 *P-Peculiarity is the absolute difference between the actual probability distribution of the categorical values and the uniform probability distribution. P-peculiarity is denoted as $pPec(a_c)$ for categorical attribute a_c, and is given by:*

$$pPec(a_c) = \sum_{x \in \mathcal{V}_{a_c}} |P(x) - \frac{1}{|\mathcal{V}_{a_c}|}|.$$

We now investigate how this measure can be normalized. We observe that the maximum deviation from the uniform distribution occurs when all categorical values except one occur exactly once. The remaining one occurs for all other entities in the table. Hence, p-peculiarity normalized to $[0, 1]$ by factor $max(pPec(a_c))$ is given by:
$(|\mathcal{V}_{a_c}| - 1) \left| \frac{1}{|\mathcal{T}|} - \frac{1}{|\mathcal{V}_{a_c}|} \right| + \left| \frac{|\mathcal{T}| - |\mathcal{V}_{a_c}| + 1}{|\mathcal{T}|} - \frac{1}{|\mathcal{V}_{a_c}|} \right|$. The normalized P-Peculiarity, $\hat{pPec}(a_c)$, is then simply given by:

$$\hat{pPec}(a_c) = \frac{\sum_{x \in \mathcal{V}_{a_c}} |P(x) - \frac{1}{|\mathcal{V}_{a_c}|}|}{max(pPec(a_c))}.$$

According to the formulation of $max(pPec(a_c))$, it is clear that $\hat{pPec}(a_c) = 1$ indicates that one specific categorical value has almost full coverage over the entities. On the other hand, $\hat{pPec}(a_c) = 0$ only for the case where each categorical value has exactly the same coverage. Both cases are considered to be not-interesting for further categorizations of an entity list. Our intuition is that $\hat{pPec}(a_c)$-value in mid range of [0,1] is considered to be interesting for

Table size	list	$p\hat{P}ec$	mIg
100	list1: { USA(90), Spain(10)}	0.82	0.98
10	list2: {USA(9), Spain(1)}	1.0	0.95

Table 6.5: Comparing $p\hat{V}ar$-value with mIg.

further categorizations of an entity list. We present the distribution of $p\hat{P}ec$-value of retrieved training samples in Figure 6.8. Following our intuition about the range of $p\hat{P}ec(a_c)$-value for interesting category, we observe that 78% of the interesting categories hold $p\hat{P}ec$-values in $[0.2, 0.8]$ (cf., Figure 6.8(a)) and 67% not-interesting categories hold $p\hat{P}ec$-values within ≤ 0.2 and ≥ 0.8 (cf., Figure 6.8(b)).

Similar to the max-info-gap, the normalizing factor used in p-peculiarity also rewards the $p\hat{P}ec$ value as table size increases. However, increasing table size rewards p-peculiarity values more prominently than the max-info-gap for the skewed distribution. For example, in Table 6.5, we can see that $p\hat{P}ec(list2) = 1$, indicating that this is not an interesting case for further categorization of the list, whereas $mIg = 0.95$, indicating that this could be an interesting case for further categorization (recall that $mIg \rightarrow 1$ indicates interesting scenarios). On the other hand, we can see that with larger list, the $p\hat{P}ec(list1)$ comes close to the mid-range of [0,1] for the same skewed distribution as list2, indicating an increasing significance for the categorization of the list. We can observe the similar characteristics by comparing the distribution of p-peculiarity and max-info-gap measures for interesting samples in Figure 6.8(a) and Figure 6.6(a) respectively—only 1% of interesting samples hold $p\hat{P}ec \geq 0.8$, whereas 31% of interesting samples hold $mIg \geq 0.8$.

6.3.3 Tailoring Support Vector Machines

After the discussion of possible features, we now describe how a classifier is trained based on different combinations of them. We opted for applying the support vector machine (SVM) approach, a widely known and well-understood concept. In the easiest case, for a balanced (roughly the same number of positive and negative samples) and linearly separable training data, a linear SVM is used to train the classifier. However, the extracted the training data from Wikipedia contain noise for various reasons, discussed earlier. Hence, we need to employ a *soft-margin classifier*, specifically, ν-SVM [SSWB00] which can detect outliers while learning the classification model from the training data. In ν-SVM, the parameter ν is tunable within $[0, 1]$. It controls the lower and upper bound on the number of misclassified samples that are allowed to use in the training phase, and thus, enables the tuning of the training error. As ν increases, the model becomes more biased and prone to underfitting the data. Moreover, the statistical measures that are used as features in this work are not always linearly separable. Multiple non-contiguous ranges of feature-values can be associated

with a specific class of samples. Hence, we employ the popular Radial Basis Function (RBF) as kernel function [SS01] that transforms our training data to higher dimensional space by using a non-linear mapping. This 'kernel' method allows classifying the training data linearly in the higher dimensional space. In the RBF kernel, a parameter γ is used to control the radius of influence of the support vectors. For example, large γ-value discards the influence of ν and cannot prevent overfitting. In Section 6.4, we will discuss how the optimal values for ν and γ are calculated to train the classifier.

In this work, we extracted 16 times more negative samples than the positive ones from Wikipedia using Algorithm 4, discussed in more detail in Section 6.4. For such an imbalance training data, the one-class SVM [SPS$^+$01] can be employed, where the classifier is trained based on the samples from a single class (either positive or negative training samples). Another option could be undersampling the data to obtain a balanced training data. In Section 6.4, we present a comparative study on the performance of the classification models which are trained by either ν-SVM on balanced samples created from original training data or one-class SVM on imbalanced original training data.

6.3.4 Evaluation Methodology

The learning model is validated in two different ways: (i) based on **held-out test data** and (ii) by means of a **user study**.

For held-out test data, the samples, extracted from Wikipedia tables using Algorithm 4, are considered as ground truth to evaluate the performance of learned classification model. We mainly use accuracy as the performance metric in this work. Additionally, we use class-specific accuracy (i.e., precision), recall, and F-measure, discussed in Section 2.3, to evaluate classification models.

Our objective is to learn a classifier that can classify the interesting categories for the categorization of an entity list. Therefore, we also validate the classification model by means of a user study. As human-perceived interestingness is not a fixed concept, choices/preferences of users can differ. The human assessors are asked to classify each test sample into one of three possible categories: **interesting**, **not-interesting**, and **not sure**.

Subsequently, we define the ground truth depending on the agreement level of the human responses. The higher the agreement level (e.g., all agree on a label), the more "obvious" the task appears, and thus, it is presumably also easier for the classifier to correctly classify it. We see later that this is indeed the fact.

Consider a total of y users that provide assessments of the test samples. Different agreement levels are considered based on majority voting: For the x/y agreement level, where $x > y/2$, one of the three possible choices is considered as ground truth for a sample if at least x users agree on that choice. The samples which are marked as *not sure* are excluded from the ground truth. For different agreement levels, class-specific accuracy/precision, recall, and F_1 measures are

used to evaluate the performance of the classification models against the ground truth generated from the user study. In order to quantify the reliability of user agreements, we use Fleiss' kappa [Fle71], discussed in Section 2.3.

6.4 Experimental Evaluation

To obtained the training samples, we have used the English version of a 2016 Wikipedia dump file. We have implemented Algorithm 4 in Java 1.8. Experiments are executed on an Intel Core i7 CPU@3GHz machine, with 16GB main memory. We use the LIBSVM [CL11] library to train the classification models. The entire process of extracting the training data from the 50.49GB (uncompressed) Wikipedia dump took around 30 minutes, where most of the time was spent on actually cleaning-up the raw dump file before Algorithm 4 was applied. We extracted a total of 2045 ranking tables from Wikipedia pages entitled "List of ...". From these tables, based on Algorithm 4, 2519 categorical attributes are labeled as "not-interesting" which are considered as negative samples and 158 categorical attributes are labeled as "interesting" which are considered as positive samples.

For the training data, 75% of the positive and negative samples are randomly selected. The remaining 25% samples from each class are considered as **held out test data**, denoted as **TestPos** and **TestNeg** respectively for the positive and the negative samples. These two test datasets are merged into a set denoted as **Test** which contains 669 samples. We retrieved significantly fewer positive samples than negative samples. To create balanced training samples, we equally divided the negative samples into ten smaller chunks and then merged each of these chunks with the positive training samples, resulting in 10 sub-training files, each containing 306 training samples. The ratio of positive and negative samples in these sub-training files are 1:1.5.

For repeatability and adoption, the labeled training data retrieved by Algorithm 4, the 10 sub-training files, and the results of the user study are publicly available under `http://dbis.informatik.uni-kl.de/catmining/`.

6.4.1 User-Study Setup

As mentioned earlier in Section 6.3.4, we set up a user study to validate the trained classifier. For this purpose, 110 randomly selected samples from Wikipedia are presented to users. The samples are displayed to a user in form of "$A(a_s) : B$". In this format, 'A' represents the title of the Wikipedia table, 'a_s' represents the subject of the table, and 'B' represents a categorical attribute associated with the entity lists in the table. Users are asked to label the samples in three categories: (i) If a user is interested to categorize the entities in table 'A' by using categorical attribute 'B', then the sample is labeled as **interesting**, (ii) If a user thinks it is not interesting to categorize the entities in table 'A' by using categorical attribute 'B', then the sample should be annotated with

not-interesting, (iii) a user can also label a sample **not sure** in case the user can not decide any of the two options before.

Overall, each question is evaluated by 9 human evaluators. Four evaluators are in fact enough to achieve a significance level[1] of $\alpha = 0.05$.

On average, an evaluator has marked 35.5% questions with interesting, 53.5% questions with not-interesting, and 20% with not sure. As the users' choices differ, we use Fleiss' kappa to understand the reliability of agreement. For each testing sample, *nine user choices* are taken for assessment. For the nine evaluators, the calculated kappa value is 0.45 and the 95% confidence interval for Kappa has a range between 0.42 and 0.48 for the collected user data. Moreover, this range of value significantly differs from zero and p-value $\approx 0 \ll 0.05$, which clearly rejects the *null hypothesis* that the agreement among users is achieved randomly.

6.4.2 Parameter Selection and Evaluation

Due to the imbalanced size of the available training samples, discussed earlier, it seems feasible to use a one-class SVM to learn a model *separately for each class*. Alternatively, we also have created the 10 balanced sub-training files from the original training sample, as mentioned above. Here, we evaluate the performance of classification models created by all feature combinations from $\mathcal{F}$, in total $2^{|\mathcal{F}|} - 1$ combinations. The classification models are created for all possible feature combinations from each sub-training file. Modifying the parameters tuning in the LIBSVM library, we implemented a grid search for ν-SVM with 5-fold cross-validation to find the optimal parameter pair (ν, γ) for each sub-training file. Finally, the classification model is learned with optimal ν and γ. According to the theoretical discussion in [CLS05], the solution of ν-SVM is only feasible for $0 \leq \nu \leq 0.77$ for our training data. In fact, in line with the theoretical study, we found that the optimal ν-value lies in $[0.41, 0.61]$ for different sub-training files with optimal $\gamma = 0.0003$. For each feature combination, the average training time of 10 sub-training files is $13.813s$.

6.4.3 Results Based on Held-Out Data

The performance of the classification models, created on sub-training files, is first evaluated on held-out test data which is created by randomly selecting 25% samples from complete samples. The held-out test data also contains 16 times more not-interesting samples than the interesting ones, reflecting the same characteristic as the training data. Therefore, a high overall accuracy on Test data does not imply that the class-specific accuracy is also high, i.e., the model performs well for both TestPos and TestNeg separately. Hence, we consider the class-specific accuracy, i.e., the precision of TestPos and TestNeg separately to evaluate the classification model, rather considering the overall accuracy of the

[1] With 9 evaluators and 3 possible answers for each task, there are 3^9 possible outcomes. Full agreement has, thus, a random chance of $3/3^9 = 0.00015$, 6/9 agreement has random chance of 0.03 for one-tail observation.

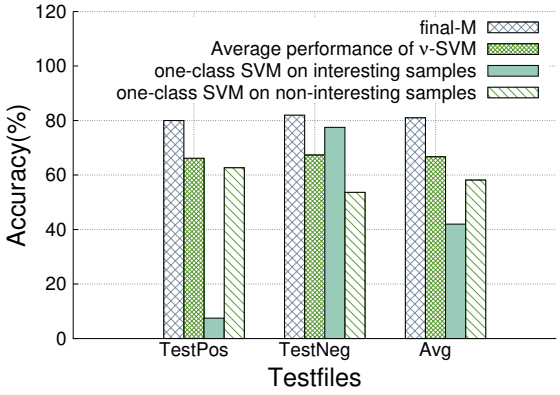


Figure 6.9: Comparison among different type of models.

classification models on Test. For each feature combination $f \in 2^{\mathcal{F}}$, we train 10 classifiers based on the 10 sub-training files and choose the one, denoted as $\textbf{best}_f$, that has minimum classification error on *both* the TestPos and TestNeg. Finally, the classification model which performs best among all best_f for feature combinations $f \in 2^{\mathcal{F}}$, is chosen as the final model, coined **final-M** in this paper. We find out that the final-M is trained using *all features except entropy and unalikeability*, reaching an accuracy of 80% and 82.003% respectively for TestPos and TestNeg.

For TestPos and TestNeg datasets separately, Figure 6.9 compares the performance among final-M, two different one-class SVM models, built on positive and negative training samples separately, and the average performance of all best_f, created on feature combinations $f \in 2^{\mathcal{F}}$. From this figure, we can observe that the performance of the classification model created from interesting training samples using one-class SVM reaches 77.46% accuracy for TestNeg. But its performance is very poor (only 7.5%) for TestPos. This model is clearly unable to detect outliers and is underfitting the data, which is unacceptable for a reasonable classifier. Though, the classification model built on not-interesting samples using one-class SVM performs consistently for both TestPos and TestNeg, the performance is inferior to the average performance of best_f created by using ν-SVM method. Finally, final-M is clearly outperforming all other models.

6.4.4 Evaluation Based on User Study

Earlier, we have discussed how we select the best model final-M from all best_f models trained for each feature combinations. The selected model is evaluated over Test data which is retrieved from Wikipedia using Algorithm 4. Hence, the validation of the final-M requires further user relevance assessments. In this section, we discuss the evaluation of the classification model based on a user study to validate our whole approach. As mentioned earlier in Section 6.3.4, we consider different levels of user agreement based on majority voting. For each x/y agreement level, we divide the ground truth into two test files. The

samples which are marked 'interesting' in the ground truth of user study with x/y agreement level, is denoted as **x/y-userPos**. On the other hand, the samples which are marked 'not-interesting' in the ground truth of x/y agreement level are denoted as **x/y-userNeg**. Here, we exclude the samples which are marked as 'not sure' from the ground truth. As the agreement level decreases, the ground truth contains more not-interesting samples than interesting ones. For the lowest agreement label, the 5/9-userNeg dataset contains almost two times more samples than 5/9-userPos dataset.

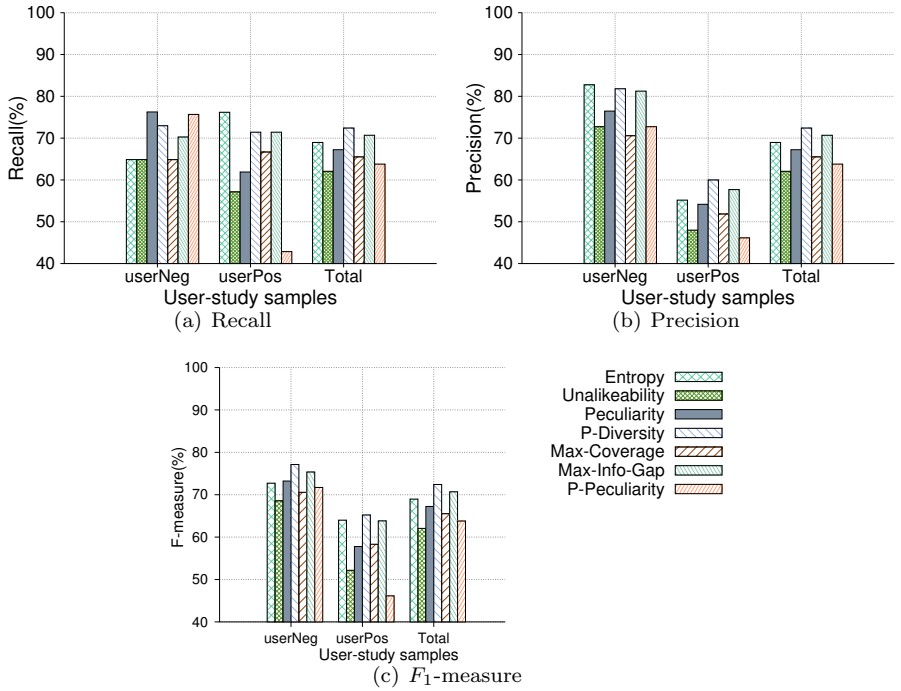
Performance of Individual Features

To understand how well each of the features, i.e., the statistical measures, can classify the data, Figure 6.10 compares the performance across the best_f classification models created over *one single feature* $f \in \mathcal{F}$. The ground truth is considered 6/9 agreement level of user assessment for this figure. We can see from the figure that the model created based on p-diversity is outperforming all the other models and reaches 72.41% overall accuracy for the test samples from the user study. Also, the model created based on Max-Info-gap is performing 2nd-best (70.69% overall accuracy). The performance of both models is consistent throughout all agreement levels, except 8/9 agreement level where the model based on entropy reaches slightly higher accuracy—the detailed study is given in Appendix A.1. Figure 6.10 also shows that the precision of all these models for 6/9-userPos is comparatively low than 6/9-userNeg. The precision improves as the agreement level increases and reaches 100% and 83.33% for 9/9-userPos and 9/9-userNeg, respectively. Comparing the F_1-measure, i.e., the harmonic mean of precision and recall, we observe that the model created using p-diversity is outperforming the other models created on single features. These results reflect our claim that the proposed measures can capture the characteristics better than the existing ones (e.g., entropy, max-coverage, etc.) to find the meaningful categorization scenarios in a human-perceived sense.

Performance of Feature Combinations

Now, we investigate the performance of the final-M which is trained using the features **peculiarity, p-diversity, max-coverage, max-info-gap, and p-peculiarity** with the model created based on the existing measures only, i.e., entropy, unalikability, peculiarity, and max-coverage, denoted as **M-Ex**. Final-M achieves **79.31**% overall accuracy, which is much higher compared to M-Ex that achieves only 67.24% for the 6/9 agreement level. Figure 6.11 shows that final-M is more robust and achieves higher F_1-measure than M-Ex based on 6/9 agreement level of the user study.

Table 6.4.4 reports on the performance of final-M for different agreement levels. Now, the higher agreement means that human evaluators had no difficulty to accomplish the classification task in a consistent way which indicates that the task is relatively easy to solve. For such presumably simple tasks, the successful



(a) Recall (b) Precision

(c) F_1-measure

Figure 6.10: Comparison of single-feature models considering 6/9 agreement level as ground truth.

classification by automated means is, thus, also more likely. The results in Table 6.4.4 are reflecting the same characteristic as the classification performance of the model increases with the increment of the agreement level.

Next, we categorize all best_f models, created earlier using all possible feature combinations, into seven groups based on the number of features used to train the model, i.e., $1 \leq |f| \leq |\mathcal{F}|$. The performance of the best performing model from each group is reported in Figure 6.12 based on 6/9 agreement level. In this figure, we also mention which features are used to create the best one within each group. For instance, the model using the features entropy, p-diversity, and max-info-gap is performing best among all the models created by combining any three features from $\mathcal{F}$.

Assessment of Main Hypothesis

Figure 6.13 presents an evaluation of Algorithm 4, respectively our main hypothesis, by comparing the retrieved labels for userPos and userNeg samples using Algorithm 4 with the human-labeled ground truth for different agreement levels. Figure 6.13 shows that the algorithm can identify the positive sample precisely (c.f., high precision on userPos). As discussed earlier in Section 6.2, due to the incompleteness of Wikipedia data, the algorithm missed out many positive samples that are marked as 'interesting' by users, which is reflected

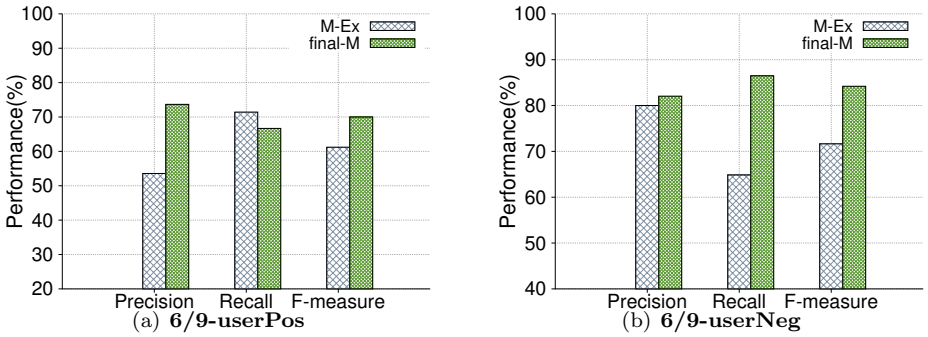


Figure 6.11: Performance of final-M vs. M-Ex.

Agreem.	**userNeg** samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	83.33	90.90	83.33	100.00	90.90	90.90
8/9	91.67	78.57	84.61	70.00	87.50	77.78	81.81
7/9	86.36	82.61	84.44	75.00	80.00	77.41	81.58
6/9	86.48	82.05	84.21	66.67	73.68	70.00	79.31
5/9	84.21	78.69	81.36	53.57	62.50	57.69	74.12

Table 6.6: Performance of final-M for different agreement levels.

by low recall on userPos shown in Figure 6.13. Figure 6.13 also shows that Algorithm 4 almost correctly identifies all samples that are marked as 'not-interesting' by users for userNeg. Moreover, final-M which is trained based on the training samples retrieved by Algorithm 4 reaches reasonable performance for both userPos and userNeg data, presented in Table 6.4.4. *These findings strongly support our working hypothesis* (cf., Section 6.2) which states that positive and negative training samples can be derived from the presence and absence of tables in Wikipedia.

Figure 6.14 shows that the models created on different features achieve almost the same classification accuracy for user-study samples according to two different ground truths: (i) the level retrieved by Algorithm 4 and (ii) 6/9 agreement level from our user study. This performance remains consistent for all other models which are not shown in the figure. This result emphasizes the robustness of the trained classification models.

6.4.5 Lesson Learned

Now, we summarize the observations from the above experimental studies in the following four lessons.

1. Our working hypothesis, which is the main idea behind Algorithm 4, is well-grounded: The positive and negative training samples in the obtained training data are generally confirmed by human evaluators.

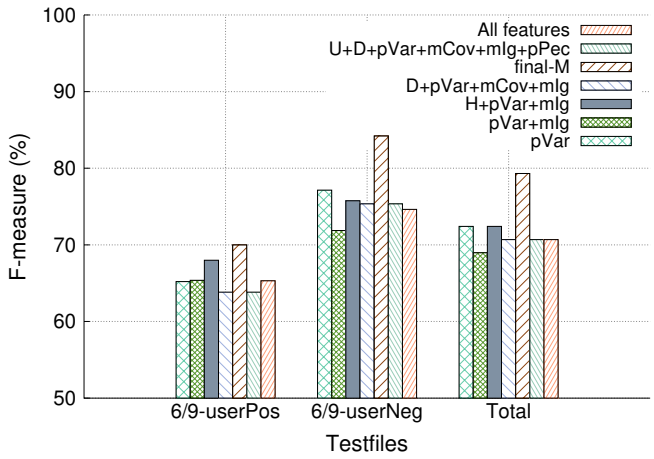


Figure 6.12: Comparison among best performing models for different numbers of used features, for 6/9 agreement level, using F_1.

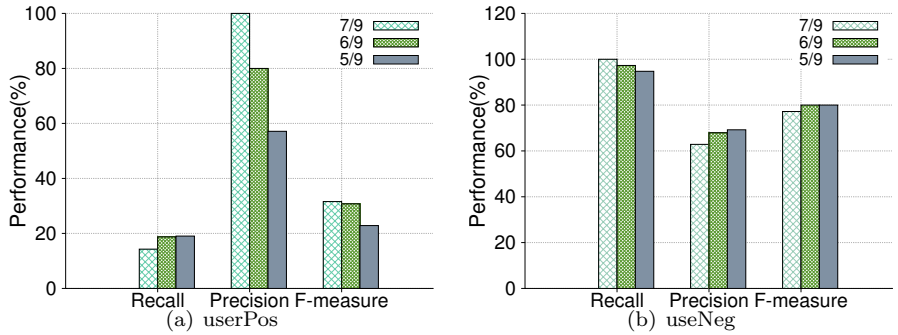


Figure 6.13: Performance of Algorithm 4.

2. Our study shows that p-diversity outperforms existing measures for the task to capture diversity in our context. The proposed measure max-info-gap performs better than max-coverage and entropy.

3. Final-M is able to achieve 79.31% overall accuracy for the ground truth given by the 6/9 user-agreement level. It uses all the features discussed in this work, except entropy and unalikeability, to train the model.

4. Final-M can accurately classify the data even when disagreement among the users' increases (i.e., the classification task gets more difficult).

6.5 Summary

In this chapter we presented a new approach to capture human interest in non-numeric categorical attributes of an entity class in a complete automated means. We made the initial hypothesis that training data can be derived from Wikipedia

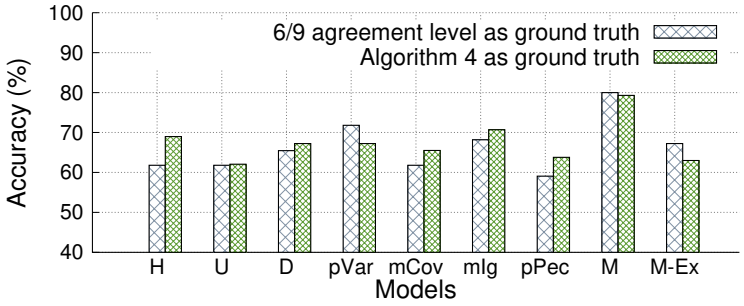


Figure 6.14: Performance of models w.r.t. different ground truths.

based on the presence or absence of specific tables. We motivated and defined three new statistical measures to capture subjective interestingness for our context of finding interesting attribute to a categorize entity list. The results of the experimental study, involving a user study, show that using features combination, a classification model can well reflect human interests on categorical attributes. Based on the user study, we also validate our proposed hypothesis. It also shows that the proposed statistical measures are more suitable to capture human interest compared to the traditional measures like information entropy.

Chapter 7

Applications of the Classification Model

This chapter is partly based on our demo publication at VLDB 2016 [PMMP16]. In this chapter, we present a brief description of two full-fledged prototypes, PANTHEON and PALEO, which allow users to explore underlying data, presented in tabular forms. Both systems adopt the proposed classification model from the previous chapter as a component, to assist the functionalities provided by the systems, as well as increase the efficiency of the framework. PANTHEON guides a user to navigate through a ranking database created automatically from a Knowledge Base. It also allows a user to issue a query about an entity type, along with a criterion (optional), to our ranking database, for extracting information about lists of entities from the given type, ranked based on different criteria. In contrast to PANTHEON, the system PALEO allows a user or data analyst to find potential queries that can generate the user-given ranked entity list from a pre-defined database.

7.1 PANTHEON

One of the challenges in handling Big Data is about making sense of large collections of complex and dynamic information. Rankings are an essential and easily comprehensible methodology to summarize the key facets of knowledge while requiring little or no understanding about the underlying data. We have developed PANTHEON, a holistic and versatile approach to compile interesting rankings from knowledge bases, without human intervention, and subsequently exploit these rankings for data exploration.

Consider a user who is traveling to the United States and wants to explore exceptional entities worth visiting. Being fascinated by city skylines, she specifies "Skyscrapers" as a domain of interest for exploring the data. PANTHEON finds the most prominent rankings within the domain, and allows her to explore the

domain starting with a ranking list, for example, PANTHEON returns a list of skyscrapers, located in the United States and ranked by their height. Now, she might want to continue browsing within the same domain of interest and learns about skyscrapers in other locations or further fine-tuned the exploration by adding additional constraint, e.g., skyscrapers in New York. Alternatively, she can also decide to change the domain and look at rankings that contain mountains in the United States, which she can add to her itinerary. PANTHEON enables such facility to explore data by transforming a knowledge base to a well-selected set of rankings, containing outstanding entities, that are deemed interesting to users. This is achieved by computing statistical properties, such as entropy, coverage, unalikeability, etc., over categorical attributes, in order to put thresholds on ranking competitiveness, and to limit the ranking sizes to the most interesting top portions.

Now, we describe briefly the framework of PANTHEON that finally builds a ranking database from a Knowledge Base, adopting the proposed classification model from the previous chapter within it.

7.1.1 System Framework

PANTHEON comprises two components. The core component creates a ranking database, holding meaningful rankings, generated from an underlying knowledge base (KB), such as Yago, Freebase, or DBpedia. The UI component allows users to explore the underlying KB by navigating through the generated rankings.

At the time of ranking generation, PANTHEON considers all facts, i.e., subject-predicate-object (SPO) triples, provided by the KB as input, via one or multiple files, or read from corresponding tables of a relational database system. To illustrate how such facts look like, the following triplet states that the entity 'Empire State Building' is 381m high.

```
Empire_State_Building => hasHeight => 381m
```

Figure 7.2 shows an overview of the system. It presents different steps and connection among them to generate the ranking database. Briefly put, the rankings are created using following steps:

1. Create ranking 'skeletons' from SPO triplets in KB using Apache Spark.
2. From the vast number of possibly ranking skeletons, find only the interesting ones using a classifier.
3. Assign a meaningful ordering criterion and direction (ASC/DSC) to the ranking skeletons.

A ranking skeleton is a group of entities retrieved by imposing a constraint (predicate=object) over SPO triplets and it does not contain information about ranking criteria. For instance, "Skyscrapers in Europe" is a ranking skeleton,

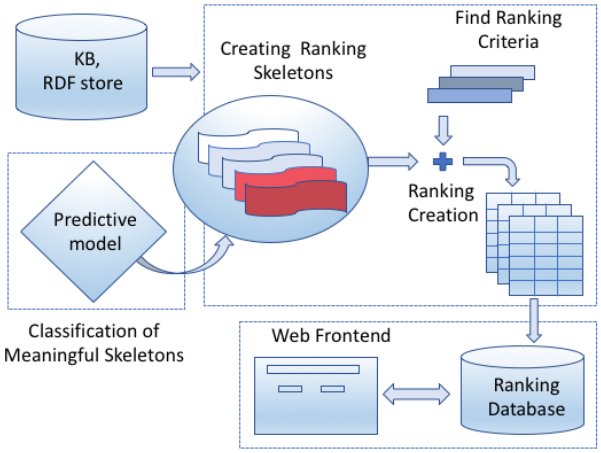


Figure 7.1: PANTHEON framework

while "Highest skyscrapers in Europe" is a ranking. One ranking skeleton may resulted in multiple rankings, using different ranking criteria. The rankings are stored in PostgreSQL with appropriate indices and data normalization, which allows interactive browsing of the rankings with low latency.

Skeleton Creation. In this work, we use Yago to generate the ranking database. Yago triplets are divided into three categories: (i) basic facts, like `Albert Einstein => hasWonPrize => Nobel`, to represent information about an entity, (ii) type facts, such as `Albert Einstein => isType => Scientist`, to group similar type of entities, and (iii) literal facts, like `Albert Einstein => bornIn => 1879`, to describe numerical information about an entity. We transform the Yago triplets into `(predicate => (object => {subject list}))` tuples, adopted from the work by [IMS13]. The type and basic facts are merged and grouped by their (predicate, object) values to create a basic set of ranking skeletons with only one constraint. For instance, in this step, a ranking skeleton like 'scientists who have won the Nobel Prize' would be created.

Finding Interesting Skeletons. Yago is a diverse, large, and popular KB that is automatically built from different data sources, e.g., Wikipedia, WordNet, and GeoNames. It covers a wide range of topics, such as geographical entities, personalities of history, movies, individuals across the science. Clearly, all ranking skeletons with single constraint, which are generated automatically in the last step from Yago, are not equally meaningful, thus, not all of them can create interesting and comprehensible rankings. At this point, it is important to realize which ranking skeleton can create a useful ranking. Otherwise, ranking creation procedure can grow exponentially, in both time and space, due to merging of multiple skeletons together to create complex ranking skeletons with multiple constraints and joining of each skeleton with multiple criteria available in Yago to create the final rankings. Following the discussion about the gen-

eral interest of humans on meaningful categories from the previous chapter, we consider the following two scenarios that can guide us to understand whether a ranking skeleton should be pruned or not.

- Depending on the domain, a ranking skeleton can be associated with a really long list of subjects, for example, list of scientists who won Nobel prize (scientists are the subjects in such Yago triplets). This is definitely a scenario where the skeleton should not be pruned, rather it can be merged with other constraints to create an interesting and more comprehensible rankings.

- A predicate can be associated with very many objects, but each (predicate, object) pair links to very few subjects.For example, skeletons for each (hasMarried, an individual) pair is generated in the first step from the Yago facts, but each of such pairs is associated with very few subjects (i.e., the number of spouse). Such scenarios are definitely not desired for further creation of a ranking and needed to be pruned.

Moreover, it is not feasible to employ human assistants to annotate the interesting scenarios. To solve this problem, PANTHEON utilize the classification model final-M, proposed in Chapter 6, that can detect the predicates that are interesting for humans to be used as a constraint (predicate=object) in rankings. For this purpose, we first need to compute different features over the frequency distribution of the values of a categorical constraint. Here, we consider (predicate, objectType) as categorical attribute. For example, if (hasWonPrize, Awards) is considered as a categorical attribute, (hasWonPrize, Nobel), (hasWonPrize, Lorentz Medal), etc., are considered as categorical values. The frequency of each instance of Awards (e.g., Nobel, Lorentz Medal) is the number of subjects associated with it. These frequency values are then used for computing the statistical measures, peculiarity, max-Coverage, max-info-gap, p-diversity, and p-peculiarity (see Section 6.3.1 and Section 6.3.2). The collected measures for a specific predicate (e.g., hasWonPrize) are used as features to fed into the classification model which predicts if the predicate is interesting/meaningful enough to create the (predicate, object) pairs. In case of positive prediction by the classification model, we keep the skeleton using single constraint, i.e., imposing (predicate = object). Additionally, but more ad-hoc, an additional pruning step is applied to remove all ranking skeletons having less than a threshold number of entities qualifying for it. In this work, we fixed this threshold to thirty. To obtain more complex rankings, two ranking skeletons are joined by combining their lists of constraints, and the associated subject list for the combined constraint is obtained by intersecting their subject lists, as discussed in [IMS13]. We continue this merging step iteratively, combining at most three (predicate, object) pairs.

Finally, we generated $2,116,942$ ranking skeletons, based on 189 predicate combinations (max. 3 predicates together). After pruning ranking skeletons, as described above, we are left with $536,982$ ranking skeletons, each containing at

least 30 entities. By using the classification model, we prune a total of 41,571 skeletons.

Ranking Criteria We consider all the literal facts in Yago as ranking criteria which are joined with the ranking skeletons to generate the actual rankings. For each pair of ranking skeleton and criterion, we find the intersected subject list and annotate the subjects with the corresponding numerical value, to create the actual ranking. Here, we ignore the subjects that do not have an associated numeric value in Yago. Finally, we rank the subjects according to the associated numerical values, where the order of the ranking (descending/ascending) is decided by analyzing the kurtosis and skewness of the distribution of numeric values, before adding the ranked entities to our ranking database.

The ranking skeletons, obtained from the previous step, are then joined with 19 distinct ranking criteria, which yields the total 339,816 rankings. Each of these rankings holds at least 30 entities with their ranking score, as per the threshold mentioned earlier. These rankings are generated over 30,145 distinct categorical constraints in the form of Predicate = Object. There are 747,147 distinct entities from various domains present in these rankings.

UI component The generated rankings are normalized into predicate, object, subject, and ranking sets and stored in a PostgreSQL database. We added several indices for faster data access, in particular, *spatial indices* are added on predicates, objects, subjects, and literals, in order to enable efficient similarity search on these attributes.

To enable searching and browsing through the ranking database, a lightweight web service is created. It consists of multiple PHP scripts, providing an API to search for predicates, objects, subjects, and rankings in the database, using keyword search. Using the API, a user can narrow down the search attribute by attribute and display intermediate results. This API is accessed by an HTML/JavaScript front end, providing a multi-step search interface to a user for finding interesting rankings. These rankings are displayed in a tabular fashion, while the system asynchronously searches for similar rankings in the background.

7.1.2 Application Scenarios

In this section, we discuss how PANTHEON allows users to explore large data by navigating through generated ranking database, without any knowledge about underlying KB.

Exploring Entities via Rankings In this application scenario, a user explore and learn about the top-performing entities from a domain of interest. In addition, while browsing the displayed rankings, by changing the constraints and ranking criteria, there is an opportunity of discovering other entities from

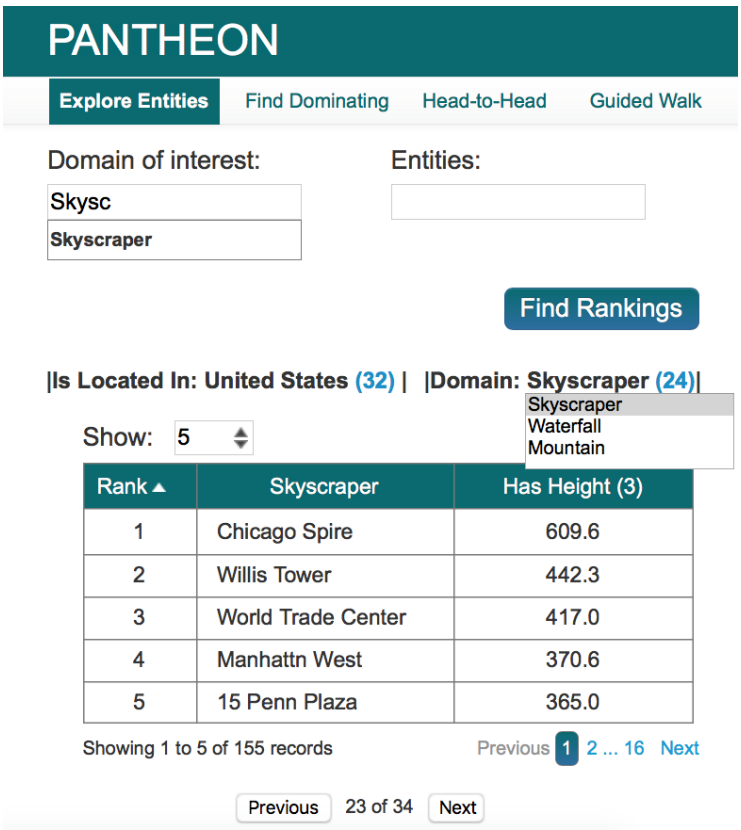


Figure 7.2: Exploration scenario in Pantheon.

a different domain, which are related to one of the initial interest. Figure 7.4 shows a screenshot demonstrating this scenario. The user can specify the domain of interest and (optionally) some constraints and get the most prominent rankings that adhere to the specifications. The system provides assistance while selecting this initial constraints.

Consider a user who is interested in skyscrapers. By specifying "Skyscrapers" as domain, she obtains 34 different rankings with different categorical constraints using PANTHEON. Figure 7.4 shows a list of skyscrapers located in the United States, ranked by their height. There are several options how the user can continue. She can continue browsing through the rankings from the domain skyscrapers by clicking the "Next" button and learn about how the entities within this domain compare with different constraints. Alternatively, perhaps she is interested in visiting the United States and wants to see what are some of the other top entities, from different domains. By clicking on the link next to the Domain, she can browse through the other domains with rankings concerning the United States. Figure 7.4 shows that there are 24 rankings with distinct domains, which are reflecting entities from the United States and can be ordered by Height. Similarly, by clicking on the link next to the ranking criteria (in this

case “Height”), the user can explore rankings with different ranking criteria. Navigating through the different domains, the user can explore other rankings, such as containing the highest waterfalls in the United States or the longest rivers in the United States, which she might wish to add to her itinerary while visiting the United States.

Random Walk through Ranked Entities In this scenario, a user can randomly walk through entities that mutually share ranking domains or some properties. In this way, a users can explore how two top entities from two very different domains can be related through a path in the knowledge graph, where an edge between the entities exists if they appear together in one ranking in our ranking database. To enable this random walk, PANTHEON allows a user to select an entity from a ranking and gives access to all the rankings where the entity appears. She can then select another entity from any one of these rankings of her interest, for exploring the graph further. For example, a user can start with the domain “linguistics”, where she selects the entity “Noam Chomsky”. This allows her to access all rankings where “Noam Chomsky” appears, such as a list of people who have won the Helmholtz Medal. In that list, the user can select the next entity “Max von Laue”, which allows her to explore the domain “physics” and find a path which connects “Noam Chomsky” to “Albert Einstein”, as both, “Max von Laue” and “Albert Einstein”, appear in the ranking of “list of Nobel laureates”.

7.2 PALEO

PALEO [PM16c, PMMP16] is a system designed for exploring databases by reverse engineering OLAP-style top-k queries. Given an input result list, PALEO is able to efficiently determine input-generating SQL queries and can additionally be relaxed to find queries that generate rankings similar to the input within a certain distance bound. Here, we introduce a scenario to explain how PALEO is useful for exploring data.

Consider a user Alice who needs to make up her mind which smartphone to buy next. Alice is favoring model X, model Y, and model Z, in this order. She is interested in finding explanatory queries and in fact populated rankings that resemble this ranking. PALEO tries to determine such queries, either explicitly reflecting Alice’s preference or delivering queries and resulting rankings that are close to her ranking. Given the structure of the queries (perhaps translated to natural language) Alice learns about the categorical constraints and ranking criteria used. Given the computed rankings, Alice can further learn about other smartphones that perform perhaps even better, depending on how much PALEO is allowed to deviate from the original input ranking. Assume PALEO returned a ranking of {X, W, Y, Z} with constraints ‘storage=16GB’ and ‘brand=Samsung’, ranked by ‘battery lifetime’. What can she learn from that and how can she proceed? She can remove the constraint on the make to

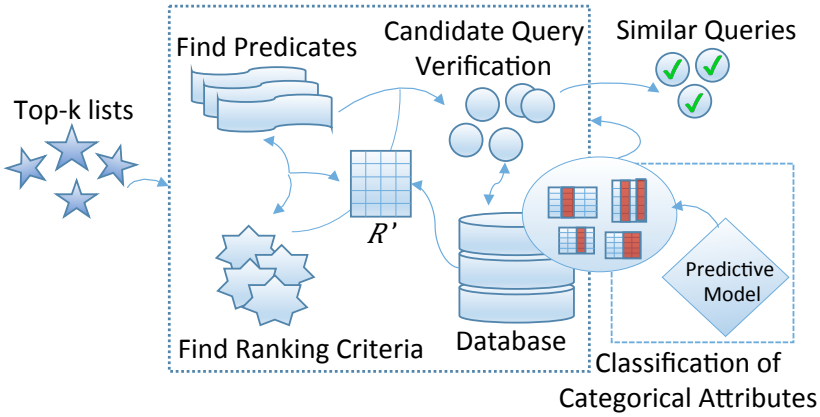


Figure 7.3: PALEO framework [PMMP16]

get additional offers, she also learned that the model W appears feasible, too. Further, she changes the ranking criteria as 'battery lifetime' is not the most decisive criterion for her anyway, can distort the ranking slightly to see how generating queries are going to differ, etc.

Now, we present the key components of PALEO framework and elaborate on how we utilize our proposed classification model from Chapter 6, in this system.

7.2.1 System framework

As user input, the system uses a top-k list L containing either a list of ranked entities only or a list of entities associated with corresponding scores. Then, given a database D with multiple relations R_i, each containing data from a single domain, PALEO efficiently and effectively determines queries Q_i that, when executed over the database, compute result lists that are **similar** to L. The found queries and corresponding top-k lists are ordered according to their similarity to the input ranking and their interestingness, in a human-perceived sense. The similarity of result rankings to the input is controlled via a user-defined similarity threshold.

The system consists of the following steps, depicted in Figure 7.3:

1. Classification of the database tables and their columns into interesting and non-interesting ones (pre-processing step).

2. Finding the meaningful predicate P in the `WHERE` clause of the reversed engineered queries Q_i from R_i .

3. Finding the ranking criterion according to which the ranked list (or a similar one) could be sorted.

4. Validation and ranking of the resulting queries and the corresponding resulting lists.

As the basis of all computation, we retrieve all tuples from the relation R, where the entity is one of the entities in the input list L into one single relation. We refer to this table as R'.

PALEO identifies a set of *candidate predicates* by using the tuples in R'. A predicate P can be a combination of multiple atomic predicate P_i, where P_i represents in the form of *attribute* = *value* (e.g., team = 'Chicago Bulls'). According to the generation of R', all distinct (attribute, value) pairs in R' are atomic candidate predicates. This would drastically reduce the performance of the system due to a radical expansion of the set of candidate predicates. To reduce the size of candidate predicates, PALEO uses two labels of pruning. The first level of pruning is done based on the preprocessing step on R', using the classification model. As discussed earlier in Chapter 6, not all the attributes in a table are meaningful to use as predicates. For example, in NBA dataset[1], last name of a coach or number of season-win by a coach is not an interesting attribute to generate a ranking list, whereas birth year of a NBA player is an interesting predicate. We use the classification model final-M, proposed in the previous chapter, to find such interesting attributes and only generate those (attribute, value) pairs, where the attribute is predicted as 'interesting' by the classification model. The second level of pruning uses user-defined distance threshold θ to prune the remaining predicates from the first level of pruning. It utilizes a bound proposed by Panev et al. [PMM16] that specifies minimum number of overlapping elements between L and entities, which are generated by applying a candidate predicate over R', are required to satisfy the given similarity threshold θ.

The candidate predicates from the previous step are combined with the criteria supported by the system to form candidate queries. The candidate queries are then executed on R' to validate if a candidate query is matching the input list. If PALEO takes scores of the ranked entities as given input, statistical methods and decision trees are used to efficiently identify the most promising column for a ranking criteria, rather combining all possible criteria with candidate predicates, in order to avoid executing all queries over R', discussed by Panev and Michel [PM16c].

In the third step, the candidate queries are further executed over original R_i. This is necessary as there are in general entities outside R' that will qualify for the ranking and might distort it. The validation is done iteratively and in each step information is gathered to potentially eliminate forthcoming queries without executing them [PM16c].

All queries that can generate the input list L do not reveal the same amount of information. Here, we again use the classification model to identify the interesting ones. PALEO assigns scores to table columns according to their interestingness, i.e., the score predicted by the classification model for the column. This score is nothing but the objective value generated by LIBSVM tools while solving the classification problem for a specific column. The higher score sig-

[1] http://www.databasebasketball.com/

nifies that the column is very likely to be part of a predicate in an interesting top-k query. Need to be mentioned here that the attribute scores are generated at the preprocessing step and is independent of the input list. The classification model final-M operates solely on five statistical measures, peculiarity, p-diversity, max-coverage, max-info-gap, and p-peculiarity, computed over the values inside a database column. This interesting score of a column is combined with the ranking similarity score, in order to reflect the tradeoff between result similarity in terms of the footrule distance to the input ranking and query-centric objectives like interestingness of column constraints.

7.2.2 Application Scenarios

PALEO enables exploration on underlying database in three different ways. Here, we very briefly describe these scenarios.

Exploring Similar Lists In this scenario, given a ranked list of entities and a similarity threshold θ, PALEO shows a set of queries together with their top-k result lists and corresponding statistics. Figure 7.4 shows a screenshot demonstrating this scenario, where a user provides a list of four NBA players (highlighted by the purple box) and a footrule distance threshold $\theta = 0.2$. Upon clicking "Find Queries" button, PALEO shows queries that can produce the input list within the distance threshold (highlighted by the red box and the green box, respectively, in Figure 7.4). Addition, PALEO shows the query execution time, and marks the interesting attributes in its predicate proposed by the classification model. A resulted query with lower Footrule distance will be displayed more prominently by default in the system, with the number of interesting categories as a second criteria to score the query. However, users can additionally change the ordering of the results by the number of interesting categories in a query's predicate. By inspecting interesting categories, users can acquire information about the categories which were used in the `WHERE` clause of the query and their distribution statistics in the database. For instance, in the example in Figure 7.4, the upper query uses only a constraint on the field position while the query below puts constraints to position as well as the league. Additional statistics of the entire process can be shown by pressing the "Under the Hood" button.

Classification of Database Columns. Understanding and exploring information in an arbitrary domain of knowledge is challenging as the complete domain usually contains a large number of entities (e.g., the NBA database contains 3924 players). A simple solution to comprehend such large data is organizing entities by ordering them using specific categories. The classification model integrated in PALEO assists a user to find interesting categories for a refined view on a subset of the entities in the example domain. For example, our model suggests that, in the NBA dataset, a refined view on the subset of players based on the position they play(ed) is more interesting than a refined

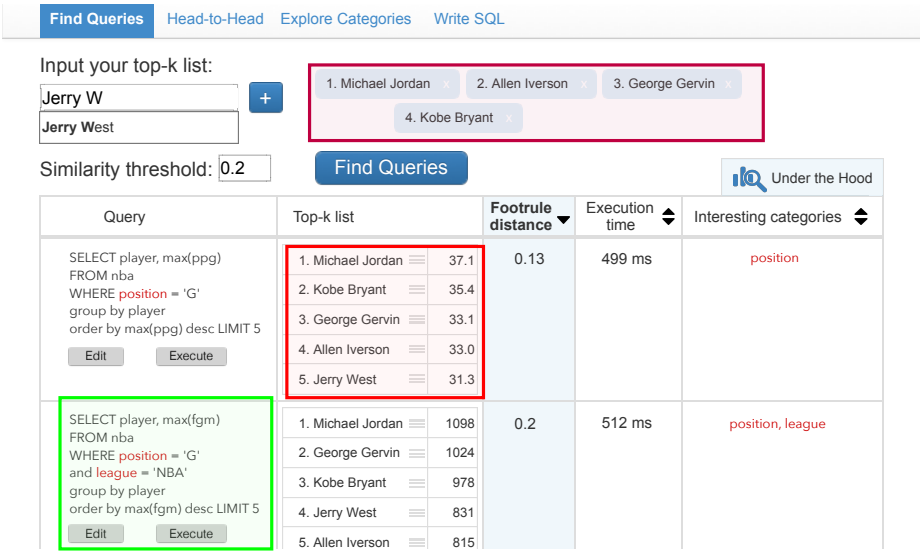


Figure 7.4: Screenshot of scenario for finding similar lists.

subset of players based on the university they went to or based on their birth date. PALEO also allows a user to have a quick overview of all tables in the database, where the table columns are scored according to the classification model and by clicking on a specific value in an interesting category, the user can add constraint to focus into a part of the data. For more insights on table columns, PALEO displays statistical characteristics on how the categorical values are distributed over the entities (Entropy, P-Diversity), how their distribution differs from the uniform distribution (P-Peculiarity), the dominating categorical values (Max-Coverage, Max-Info-Gap), etc.

Head-to-head Comparison of Entities. PALEO also allows a user to compare specific entities of their choice by comparing scoring functions and categories. In a head-to-head entity comparison, the performance of entities is agnostic to all other entities in the domain except the ones provided as input. Thus, users will also be able to compare entities that normally do not belong in the same league, i.e., one is ranked significantly higher than another, and therefore, are not together in any top-k scenario. PALEO provides an option of exploring these entities and shows how they can be compared against each other by ignoring the other entities that could (potentially) appear between them in the ranking.

Chapter 8

Conclusions and Outlook

Overall, in this thesis, we addressed two related but independent research problems, efficient similarity search over rankings and the identification of meaningful categories for an entity lists, for the purpose of exploring ranked entities or tables.

We proposed four query-driven Locality Sensitive Hashing schemes for performing efficient similarity search over ranked entities. Exploiting properties of the proposed LSH families, we determined a tighter, yet probabilistic, bound on the number of index accesses than the bound provided by the popular prefix-filtering method, for the query processing over pairwise and triple indices, which are different variations of inverted indices. We also devised a strategy of selecting the elements from a query, based on their position in the query ranking, for probing an inverted index during query processing, in order to increase the precision of the similarity search. We presented a detailed study on the efficiency of our proposed schemes using two real-world ranking datasets. Further, in order to optimize the space requirement for maintaining the index structures, used in proposed Query-driven LSH, we discussed three pruning methods and analyzed their effect on the quality of the search results. Based on this analysis, we formalized an optimization problem to find the optimal pruning factor to prune the indices, ensuring a user-given recall requirement. The proposed pruning approach is generic and can also be applied to an inverted index. We validated our optimization problem using two case studies, over a ranking database and a collection of sets.

In the second part of the thesis, we presented a completely automated framework to train a classifier that can identify the meaningful categories, in a human-perceived sense, for categorizing the entities in a table, and thus, assist users to explore entities of their interest. Here, we proposed three objective measures of interestingness, capturing different characteristics of categorical data. Using the proposed measures and two existing measures as features, we trained a classifier by using ν-SVM approach from the training data that are collected from Wikipedia tables, based on the proposed hypothesis. We validated the proposed

hypothesis and the performance of the trained classification model with the help of user assessments on a test dataset. We also reported a detailed study on how well the individual feature, i.e., the proposed and existing objective measures, is capable of characterizing the interestingness of categorical data.

Future Directions

In the problem of similarity search over rankings, we addressed the effect of the ranking position of query elements in the efficiency of the similarity search. Later, from the experimental study on real-world datasets, we observed that the distribution of query elements in ranking dataset also affects the efficiency of the proposed schemes. Here, the future direction could be incorporating the data distribution to our query-driven LSH schemes to increase the performance of the query-driven LSH schemes, particularly, in case of the skewed data distribution. Next, we discussed the optimization problem to find the optimal pruning factor to prune an index, which uses the cost for querying the index as one of the constraints. There, we assumed that the data are uniformly distributed over posting lists to find the cost for a query. The cost model can be further extended according to the actual data distribution to make the optimization problem more accurate and data-driven.

To train the proposed classifier for finding meaningful categories, we harnessed Wikipedia tables to extract the training data. There, we used a simple extraction method to avoid noise in the generation of training data. As a result, we retrieved fewer training samples to train the classifier. Hence, the open possibilities are—using complex extracting methods to retrieve the table information more accurately from Wikipedia, extracting tables beyond Wikipedia, or using query logs to capture general interests of users on categorical attributes, in order to increase the sample size. Also, depending on the success of retrieving larger training samples, we can integrate deep learning methods to extract features automatically from the sample for building the classifier.

Appendix A

Appendix

A.1 Performance of Single Feature

Here, we present the performance of the classification model, based on single feature for all different agreement levels.

Agreem.	**userNeg** samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	83.33	90.90	83.33	100.00	90.90	90.91
8/9	83.33	90.90	86.95	90.00	81.81	85.71	86.36
7/9	72.73	84.21	78.05	81.25	68.42	74.28	76.32
6/9	64.86	82.76	72.73	76.19	55.17	64.00	68.97
5/9	64.91	78.72	71.15	64.28	47.36	54.54	64.71

Table A.1: Performance of entropy for different agreement levels.

Agreem.	**userNeg** samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	71.43	83.33	66.66	100.00	80.00	81.82
8/9	83.33	76.92	80.00	70.00	77.77	73.68	77.27
7/9	72.73	76.19	74.42	68.75	64.70	66.66	71.05
6/9	64.86	77.42	70.59	66.66	51.85	58.33	65.52
5/9	63.16	76.60	69.23	60.71	44.73	51.51	62.35

Table A.2: Performance of max-coverage for different agreement levels.

Agreem.	**userNeg** samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	71.43	83.33	66.66	100.00	80.00	81.82
8/9	83.33	71.42	76.92	60.00	75.00	66.66	72.72
7/9	72.73	69.57	71.11	56.25	60.00	58.06	65.79
6/9	64.86	72.73	68.57	57.14	48.00	52.17	62.07
5/9	63.16	75.00	68.57	57.14	43.24	49.23	61.18

Table A.3: Performance of unalikeability for different agreement levels.

Agreem.	**userNeg** samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	71.43	83.33	66.66	100	80.00	81.82
8/9	91.66	78.57	84.61	70.00	87.50	77.78	81.81
7/9	77.27	73.91	75.56	62.50	66.66	64.51	71.05
6/9	70.27	76.47	73.24	61.90	54.16	57.77	67.24
5/9	68.42	76.47	72.22	57.14	47.05	51.61	64.71

Table A.4: Performance of peculiarity for different agreement levels.

Agreem.	**userNeg** Samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	83.33	90.90	83.33	100.00	90.90	90.91
8/9	83.33	90.90	86.95	90.00	81.81	85.71	86.36
7/9	72.73	84.21	78.05	81.25	68.42	74.28	76.32
6/9	70.27	81.25	75.36	71.42	57.69	63.83	70.69
5/9	71.93	78.85	75.23	60.71	51.51	55.73	68.24

Table A.5: Performance of max-info-gap for different agreement levels.

Agreem.	**userNeg** samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	83.33	90.90	83.33	100.00	90.90	90.90
8/9	83.33	83.33	83.33	80.00	80.00	80.00	81.81
7/9	77.27	80.95	79.07	75.00	70.58	72.72	76.32
6/9	72.97	81.82	77.14	71.42	60.00	65.21	72.41
5/9	75.44	78.18	76.79	57.14	53.33	55.17	69.41

Table A.6: Performance of p-diversity for different agreement levels.

Agreem.	**userNeg** samples			**userPos** samples			
level	Rec.	Prec.	F_1	Rec.	Prec.	F_1	Accuracy
9/9	100.00	71.43	83.33	66.66	100.00	80.00	81.82
8/9	91.67	68.75	78.57	50.00	81.81	62.50	72.72
7/9	77.27	65.38	70.83	43.75	58.33	50.00	63.16
6/9	75.68	70.00	72.73	42.85	50.00	46.15	63.79
5/9	70.18	70.18	70.18	39.28	39.28	39.28	60.00

Table A.7: Performance of p-peculiarity for different agreement levels.

A.2 User Study Samples

Table	Subject of the Table	Attribute to categorize
List of the busiest airports in Romania	airports	Code(IATA/ICAO)
List of tallest buildings in Washington, D.C.	Name of tallest buildings	Floors
List of the busiest airports in Japan	airports	IATA/ICAO
List of Knight's Cross of the Iron Cross recipients of the Fallschirmj??ger	Name of Knight's Cross of the Iron Cross recipients of the Fallschirmj??ger	Role and unit1
List of FIFA World Cup goalscorers	Player of FIFA World Cup	Goal average
List of the busiest airports in Japan	airports	Cityserved
List of Canadian supercentenarians	Name of Canadian supercentenarians	Death date
List of FC Seoul records and statistics	Name of FC Seoul records and statistics	Clean Sheets per Match
List of continents by population	Region of continents	Per Annum Growth Rate2010-2013
List of Pakistani films of 2010	Title of Pakistani films of 2010	Verdict
List of South African airports by passenger movements	airports	Code(IATA/ICAO)
List of mountain peaks of Central America	Mountain Peak of mountain peaks of Central America	Nation
List of mountain peaks of Central America	Mountain Peak of mountain peaks of Central America	Nation
List of National Basketball Association career playoff rebounding leaders	Player of National Basketball Association career playoff rebounding leaders	Team(s) played for (years)
Sunday Times Rich List 2011	name	Citizenship
List of highest-grossing Bollywood films	Movie of highest-grossing Bollywood films	Net Gross
List of prefecture-level cities by GDP	cities	Provinces
List of the busiest airports in Romania	airports	Code(IATA/ICAO)
List of tallest buildings in Ukraine	Name of tallest buildings	Status
List of Finnish supercentenarians	Name of Finnish supercentenarians	Region or country of birth
List of places in Queensland by population	Urban Centre of places	Region
List of newspapers in Canada by circulation	newspapers	Weekly Circulation 2008
List of cities in Egypt	Name of cities	Census 1986
List of tallest buildings in Providence	Name of tallest buildings	Coordinates

Table	Subject of the Table	Attribute to categorize
List of the busiest airports in the Nordic countries	Countries of the busiest airports	Airport(s) included
List of tallest residential buildings in the world	Name of tallest residential buildings	Country
List of supercentenarians from the United States	Name of supercentenarians from the United States	Sex
List of mountains in Taiwan	Name of mountains	Location
List of tallest buildings in Spain	Name of tallest buildings	Floors
List of tallest buildings in Saudi Arabia	Name of tallest buildings	Floors
List of 2004 box office number-one films in the United States	Title of 2004 box office number-one films	Director(s)
List of Jamaican supercentenarians	Name of Jamaican supercentenarians	Death date
List of best-selling albums by country	Year of best-selling albums	Artist
List of tallest buildings in Pakistan	Name of tallest buildings	City
List of airports in Spain	airports	Code
List of the 100 largest cities and towns in Canada by area	Municipality of the 100 largest cities and towns	Status
List of Major League Baseball longest winning streaks	Game of Major League Baseball longest winning streaks	Score
List of cities in Germany by population	City of cities	Growth
List of multiple Olympic medalists in one event	Athlete of multiple Olympic medalists	Sport
List of airports in Ukraine	airports	City
List of multiple Olympic gold medalists at a single Games	Athlete of multiple Olympic gold medalists at a single Games	Nation
List of Major League Baseball longest losing streaks	Game of Major League Baseball longest losing streaks	Opponent
List of top association football goal scorers by country	Player of top association football goal scorers	Country
List of Paralympic Games host cities	Countries of Paralympic Games host cities	Continent

Table	Subject of the Table	Attribute to categorize
List of the world's largest cruise ships	ships	Beam
List of supercentenarians from the United States	Name of supercentenarians from the United States	Birth date
List of Pakistani films of 2013	Title of Pakistani films of 2013	Domestic gross
List of Bollywood films of 2013	films	India
List of Australian rugby union stadiums by capacity	Stadium of Australian rugby union stadiums	City
List of African stadiums by capacity	stadiums	Home Team/s
List of tallest buildings in Cincinnati	Name of tallest buildings	Reference
Ranked list of Mexican states	states	Comparable country
List of tallest buildings in Slovenia	Name of tallest buildings	Location
List of tallest residential buildings in the world	Name of tallest residential buildings	Country
List of the busiest airports in Canada	Airport	Serves
List of U.S. cities with significant Korean-American populations	cities	Percentage
List of newspapers in Canada by circulation	newspapers	Weekly Circulation 2009
List of mountains of Switzerland above 3000 m	mountains	Range
List of accidents and incidents involving airliners in the United States	Date of accidents and incidents involving airliners	Article
List of shopping centres in Australia by size	centres	Department Stores
List of roller coaster rankings	Name of roller coaster	Record held
List of North American stadiums by capacity	stadiums	Home Team(s)
List of highest-grossing concert tours	Actual gross of highest-grossing concert tours	Artist
List of U.S. states by electricity production from renewable sources	Hydropower of U.S. states	State
List of Tamil films of 2010	Movie of Tamil films of 2010	Production
List of richest Cypriots	Name of richest Cypriots	Fortune
List of 2002 box office number-one films in the United States	Title of 2002 box office number-one films	Director(s)
List of Total Drama Island episodes	Name of Total Drama Island episodes	Team

Table	Subject of the Table	Attribute to categorize
List of Major League Baseball longest winning streaks	Total gam of Major League Baseball longest winning streaks	Team
List of Major League Baseball longest losing streaks	Game of Major League Baseball longest losing streaks	Opponent
List of Tamil films of 2009	Title of Tamil films of 2009	Cast
List of mountains in Iran	Photograph of mountains	Mountain
List of public corporations by market capitalization	Name of public corporations	Headquarters
List of tallest buildings in Osaka Prefecture	Name of tallest buildings	Coordinates
List of tallest buildings in Bangladesh	Name of tallest buildings	City
List of U.S. cities with significant Chinese-American populations	Borough of U.S. cities with significant Chinese-American populations	City
List of multiple Olympic medalists in one event	Athlete of multiple Olympic medalists	Event
List of the busiest airports in the Nordic countries	Countries of the busiest airports	Metropolitan Area
List of tallest structures in Serbia	Image of tallest structures	Floors
List of tallest buildings in Iowa	Picture of tallest buildings	Primary Purpose
List of mountains in Taiwan	Name of mountains	Location
List of Finnish supercentenarians	Name of Finnish supercentenarians	Region or country of birth
List of National Basketball Association career playoff free throw scoring leaders	Player of National Basketball Association	Team(s) played for (years)1
List of oldest and youngest Academy Award winners and nominees	Age of oldest and youngest Academy Award winners and nominees	Date of Nomination
List of tallest buildings in New Hampshire	Picture of tallest buildings	Name
List of states and union territories of India by population	State or union territories of states and union territories of India	Area4
List of tallest buildings in Europe by year	buildings	City
List of Knight's Cross of the Iron Cross with Oak Leaves recipients (1942)	Service of Knight's Cross of the Iron Cross with Oak Leaves recipients (1942)	Name
List of settlements on the island of Ireland by population	settlements	Province

Table	Subject of the Table	Attribute to categorize
List of tallest destroyed buildings and structures in the United Kingdom	Name of tallest destroyed buildings and structures	Fate
List of Gold Coast Football Club records	Score of Gold Coast Football Club records	Opponent
List of college field hockey coaches with 250 wins	Name of college field hockey coaches with 250 wins	Pct.
List of longest streams of Idaho	Name of longest streams of Idaho	Length in Idaho2
List of cities and towns in Armenia	cities	Province
List of cities in Oregon	cities	Population(2013 est.)1
List of Tamil films of 2010	Movie of Tamil films of 2010	Studio
List of cities in Swaziland	cities	District
List of mountains of Bangladesh	Peak Nam of mountains of Bangladesh	Location
List of ice hockey arenas by capacity	arenas	Country
List of the busiest airports in Ireland	Province of the busiest airports	County
List of college football stadium video boards	University of college football stadium video boards	Stadium
List of best-selling albums in South Korea	Year of best-selling albums	Title
List of Tamil films of 2009	Title of Tamil films of 2009	Production
List of cities in Germany by population	City of cities	State(Bundesland)
List of tallest buildings in Pennsylvania	Name of tallest buildings	City
List of tallest buildings and structures in Liverpool	Name (alternate names) of tallest buildings and structures	Coordinates
List of islands of Scotland	islands	Local Authority
List of record home attendances of English football clubs	clubs	Stadium
List of tallest buildings in the world	Tallest buildings	Country
List of countries by sovereign wealth funds	countries	Funds

Bibliography

[AGHI09] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In Ricardo A. Baeza-Yates, Paolo Boldi, Berthier A. Ribeiro-Neto, and Berkant Barla Cambazoglu, editors, *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 5–14. ACM, 2009.

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[AIP+12] Sameer Agarwal, Anand P. Iyer, Aurojit Panda, Samuel Madden, Barzan Mozafari, and Ion Stoica. Blink and it's done: Interactive queries on very large data. *Proc. VLDB Endow.*, 5(12):1902–1905, August 2012.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pages 207–216. ACM Press, 1993.

[AKJ04] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Machine Learning: ECML 2004, 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004, Proceedings*, volume 3201 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2004.

[AM06] Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. In Efthimis N. Efthimiadis, Susan T. Dumais, David Hawking, and Kalervo Järvelin, editors, *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seat-*

tle, Washington, USA, August 6-11, 2006, pages 372–379. ACM, 2006.

[AOU12] Ismail Sengör Altingövde, Rifat Ozcan, and Özgür Ulusoy. Static index pruning in web search engines: Combining term and document popularities with query views. *ACM Trans. Inf. Syst.*, 30(1):2:1–2:28, 2012.

[APPK08] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen, editors, *ICDE*, pages 327–336. IEEE Computer Society, 2008.

[BC06] Stefan Büttcher and Charles L. A. Clarke. A document-centric approach to static index pruning in text retrieval systems. In Philip S. Yu, Vassilis J. Tsotras, Edward A. Fox, and Bing Liu, editors, *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6-11, 2006*, pages 182–189. ACM, 2006.

[BCG05] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 651–660. ACM, 2005.

[BCH+03] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Y. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pages 426–434. ACM, 2003.

[Ben90] Jon Louis Bentley. K-d trees for semidynamic point sets. In Raimund Seidel, editor, *Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkeley, CA, USA, June 6-8, 1990*, pages 187–197. ACM, 1990.

[BFL+10] Michal Batko, Fabrizio Falchi, Claudio Lucchese, David Novak, Raffaele Perego, Fausto Rabitti, Jan Sedmidubský, and Pavel Zezula. Building a web-scale image similarity search system. *Multimedia Tools Appl.*, 47(3):599–629, 2010.

[Bie11] Tijl De Bie. An information theoretic framework for data mining. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 564–572. ACM, 2011.

[Bie13] Tijl De Bie. Subjective interestingness in exploratory data mining. In *Advances in Intelligent Data Analysis XII - 12th International Symposium, IDA 2013, London, UK, October 17-19, 2013. Proceedings*, pages 19–31, 2013.

[BKS10] Tijl De Bie, Kleanthis-Nikolaos Kontonasios, and Eirini Spyropoulou. A framework for mining interesting pattern sets. *SIGKDD Explorations*, 12(2):92–100, 2010.

[BLWJ15] Lidong Bing, Wai Lam, Tak-Lam Wong, and Shoaib Jameel. Web query reformulation via joint modeling of latent topic dependency and term context. *ACM Trans. Inf. Syst.*, 33(2):6:1–6:38, 2015.

[BML06] Judit Bar-Ilan, Mazlita Mat-Hassan, and Mark Levene. Methods for comparing rankings of search engine results. *Computer Networks*, 50(10):1448–1463, 2006.

[BND13] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Methods for exploring and mining tables on wikipedia. In Duen Horng Chau, Jilles Vreeken, Matthijs van Leeuwen, and Christos Faloutsos, editors, *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics, IDEA@KDD 2013, Chicago, Illinois, USA, August 11, 2013*, pages 18–26. ACM, 2013.

[Bro97] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–29, Washington, DC, USA, 1997. IEEE Computer Society.

[Car09] Ben Carterette. On rank correlation and the distance between rankings. In James Allan, Javed A. Aslam, Mark Sanderson, ChengXiang Zhai, and Justin Zobel, editors, *ACM, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 436–443. ACM, 2009.

[CFWB17] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1870–1879. Association for Computational Linguistics, 2017.

[CGK06] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 5. IEEE Computer Society, 2006.

[Cha02] Moses Charikar. Similarity estimation techniques from rounding algorithms. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 380–388. ACM, 2002.

[CHC+10] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490, 2010.

[CHW+08] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CLK+16] Fernando Chirigati, Jialu Liu, Flip Korn, You Wu, Cong Yu, and Hao Zhang. Knowledge exploration using tables on the web. *PVLDB*, 10(3):193–204, 2016.

[CLS05] Pai-H. Chen, Chih-J. Lin, and Bernhard Schölkopf. A tutorial on ν-support vector machines, 2005.

[CMZG09] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. Expected reciprocal rank for graded relevance. In David Wai-Lok Cheung, Il-Yeol Song, Wesley W. Chu, Xiaohua Hu, and Jimmy J. Lin, editors, *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 621–630. ACM, 2009.

[CP11] Eric Crestan and Patrick Pantel. Web-scale table census and classification. In Irwin King, Wolfgang Nejdl, and Hang Li, editors, *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pages 545–554. ACM, 2011.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.

[CS88] W. Bruce Croft and Pasquale Savino. Implementing ranking strategies using text signatures. *ACM Trans. Inf. Syst.*, 6(1):42–62, 1988.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[DC12] Van Dang and W. Bruce Croft. Diversity by proportionality: an election-based approach to search result diversification. In William R. Hersh, Jamie Callan, Yoelle Maarek, and Mark Sanderson, editors, *The 35th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '12, Portland, OR, USA, August 12-16, 2012*, pages 65–74. ACM, 2012.

[DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In Eric A. Brewer and Peter Chen, editors, *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150. USENIX Association, 2004.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In Jack Snoeyink and Jean-Daniel Boissonnat, editors, *Symposium on Computational Geometry*, pages 253–262. ACM, 2004.

[DKNS01] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001.

[dMdSF+05] Edleno Silva de Moura, Célia Francisca dos Santos, Daniel R. Fernandes, Altigran Soares da Silva, Pável Calado, and Mario A. Nascimento. Improving web search efficiency via a locality based static pruning method. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 235–244. ACM, 2005.

[DP13] Marina Drosou and Evaggelia Pitoura. Ymaldb: exploring relational databases via result-driven recommendations. *VLDB J.*, 22(6):849–874, 2013.

[DPD14] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 517–528. ACM, 2014.

[DRM+08] Debabrata Dash, Jun Rao, Nimrod Megiddo, Anastasia Ailamaki, and Guy M. Lohman. Dynamic faceted search for discovery-driven analysis. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, *Proceedings of the 17th*

ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008, pages 3–12. ACM, 2008.

[DS08] Josep Domingo-Ferrer and Agusti Solanas. A measure of variance for hierarchical nominal attributes. *Inf. Sci.*, 178(24):4644–4655, 2008.

[DWJ+08] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling LSH for performance tuning. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 669–678. ACM, 2008.

[FCS17] Nausheen Fatma, Manoj Kumar Chinnakotla, and Manish Shrivastava. The unusual suspects: Deep learning based mining of interesting entity trivia from knowledge graphs. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 1107–1113. AAAI Press, 2017.

[FKS03] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003.

[Fle71] Joseph L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

[GCB+97] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

[GFFN12] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 541–552. ACM, 2012.

[GH06] Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Comput. Surv.*, 38(3), 2006.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In Malcolm P. Atkinson,

Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB*, pages 518–529. Morgan Kaufmann, 1999.

[GJLO14] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. DSH: data sensitive hashing for high-dimensional k-nnsearch. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1127–1138. ACM, 2014.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950.

[HFH+09] Mark A. Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[HFZ+15] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *PVLDB*, 9(1):1–12, 2015.

[HH01] Robert J. Hilderman and Howard J. Hamilton. Evaluation of interestingness measures for ranking discovered knowledge. In David Wai-Lok Cheung, Graham J. Williams, and Qing Li, editors, *Knowledge Discovery and Data Mining - PAKDD 2001, 5th Pacific-Asia Conference, Hong Kong, China, April 16-18, 2001, Proceedings*, volume 2035 of *Lecture Notes in Computer Science*, pages 247–259. Springer, 2001.

[HH14] Sascha Henzgen and Eyke Hüllermeier. Mining rank data. In *Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, pages 123–134, 2014.

[HM03] Sven Helmer and Guido Moerkotte. A performance study of four index structures for set-valued attributes of low cardinality. *VLDB J.*, 12(3):244–261, 2003.

[HSBW13] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Jeffrey Scott Vitter, editor, *STOC*, pages 604–613. ACM, 1998.

[IMS13] Evica Ilieva, Sebastian Michel, and Aleksandar Stupar. The essence of knowledge (bases) through entity rankings. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1537–1540. ACM, 2013.

[IRW16] Yusra Ibrahim, Mirek Riedewald, and Gerhard Weikum. Making sense of entities and quantities in web tables. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 1703–1712. ACM, 2016.

[Jac12] Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, February 1912.

[JASG08] Herve Jegou, Laurent Amsaleg, Cordelia Schmid, and Patrick Gros. Query adaptative locality sensitive hashing. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2008, March 30 - April 4, 2008, Caesars Palace, Las Vegas, Nevada, USA*, pages 825–828. IEEE, 2008.

[JB08] Alexis Joly and Olivier Buisson. A posteriori multi-probe locality sensitive hashing. In Abdulmotaleb El-Saddik, Son Vuong, Carsten Griwodz, Alberto Del Bimbo, K. Selçuk Candan, and Alejandro Jaimes, editors, *Proceedings of the 16th International Conference on Multimedia 2008, Vancouver, British Columbia, Canada, October 26-31, 2008*, pages 209–218. ACM, 2008.

[JGP16] Manas Joglekar, Hector Garcia-Molina, and Aditya G. Parameswaran. Interactive data exploration with smart drill-down. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 906–917. IEEE Computer Society, 2016.

[JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[JRS16] Wei Jiang, Juan Rodriguez, and Torsten Suel. Improved methods for static index pruning. In James Joshi, George Karypis, Ling Liu, Xiaohua Hu, Ronay Ak, Yinglong Xia, Weijia Xu, Aki-Hiro Sato, Sudarsan Rachuri, Lyle H. Ungar, Philip S. Yu, Rama

Govindaraju, and Toyotaro Suzumura, editors, *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pages 686–695. IEEE, 2016.

[Ken38] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[KJTN14] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. Distributed and interactive cube exploration. In Isabel F. Cruz, Elena Ferrari, Yufei Tao, Elisa Bertino, and Goce Trajcevski, editors, *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 472–483. IEEE Computer Society, 2014.

[KP07] Gary D. Kader and Mike Perry. Variability for categorical variables. *Journal of Statistics Education*, 15(2), 2007.

[KS98] Norio Katayama and Shin'ichi Satoh. Sr-tree: An index structure for nearest-neighbor searching of high-dimensional point data. *Systems and Computers in Japan*, 29(6):59–73, 1998.

[KV10] Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 571–580. ACM, 2010.

[LCH+14] Yingfan Liu, Jiangtao Cui, Zi Huang, Hui Li, and Heng Tao Shen. SK-LSH: an efficient index structure for approximate nearest neighbor search. *PVLDB*, 7(9):745–756, 2014.

[LHY+08] Xiaolei Li, Jiawei Han, Zhijun Yin, Jae-Gil Lee, and Yizhou Sun. Sampling cube: a framework for statistical olap over sampling data. In Jason Tsong-Li Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 779–790. ACM, 2008.

[LJW+07] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, editors, *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 950–961. ACM, 2007.

[LSC10] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3(1):1338–1347, 2010.

[MAM15] Evica Milchevski, Avishek Anand, and Sebastian Michel. The sweet spot between inverted indices and metric-space indexing for top-k-list similarity search. In Gustavo Alonso, Floris Geerts, Lucian Popa, Pablo Barceló, Jens Teubner, Martín Ugarte, Jan Van den Bussche, and Jan Paredaens, editors, *EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 253–264. OpenProceedings.org, 2015.

[MBJ99] Katharina Morik, Peter Brockhausen, and Thorsten Joachims. Combining statistical learning with a knowledge-based approach - A case study in intensive care monitoring. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 268–277. Morgan Kaufmann, 1999.

[MW08] David N. Milne and Ian H. Witten. Learning to link with wikipedia. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 509–518. ACM, 2008.

[MY01] Larry M. Manevitz and Malik Yousef. One-class svms for document classification. *Journal of Machine Learning Research*, 2:139–154, 2001.

[Ngu09] Linh Thai Nguyen. Static index pruning for information retrieval systems: A posting-based approach. In *7th workshop on large-scale distributed systems for information retrieval (LSDS-IR'09)*, 2009.

[ODD06] Eyal Oren, Renaud Delbru, and Stefan Decker. Extending faceted navigation for RDF data. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 559–572. Springer, 2006.

[Pan06] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 1186–1195. ACM Press, 2006.

[PM14] Koninika Pal and Sebastian Michel. An LSH index for computing kendall's tau over top-k lists. In *Proceedings of the 17th International Workshop on the Web and Databases, (WebDB), co-located with SIGMOD, Snowbird, UT, USA, June 22*, 2014.

[PM16a] Koninika Pal and Sebastian Michel. A data mining approach to choosing categorical attributes for ranked lists. In Evaggelia Pitoura, Sofian Maabout, Georgia Koutrika, Amélie Marian, Letizia Tanca, Ioana Manolescu, and Kostas Stefanidis, editors, *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 664–665. OpenProceedings.org, 2016.

[PM16b] Koninika Pal and Sebastian Michel. Efficient similarity search across top-k lists under the kendall's tau distance. In Peter Baumann, Ioana Manolescu-Goujot, Luca Trani, Yannis E. Ioannidis, Gergely Gábor Barnaföldi, László Dobos, and Evelin Bányai, editors, *Proceedings of the 28th International Conference on Scientific and Statistical Database Management, SSDBM 2016, Budapest, Hungary, July 18-20, 2016*, pages 6:1–6:12. ACM, 2016.

[PM16c] Kiril Panev and Sebastian Michel. Reverse engineering top-k database queries with PALEO. In Evaggelia Pitoura, Sofian Maabout, Georgia Koutrika, Amélie Marian, Letizia Tanca, Ioana Manolescu, and Kostas Stefanidis, editors, *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 113–124. OpenProceedings.org, 2016.

[PM17] Koninika Pal and Sebastian Michel. LSH-based probabilistic pruning of inverted indices for sets and ranked lists. In Alexandra Meliou and Pierre Senellart, editors, *Proceedings of the 20th International Workshop on the Web and Databases, WebDB 2017, Chicago, IL, USA, May 14-19, 2017*, pages 23–28. ACM, 2017.

[PM18] Koninika Pal and Sebastian Michel. Learning interesting attributes for automated data categorization. In Dimitris Sacharidis, Johann Gamper, and Michael H. Böhlen, editors, *Proceedings of the 30th International Conference on Scientific and Statistical Database Management, SSDBM 2018, Bozen-Bolzano, Italy, July 09-11, 2018*, pages 9:1–9:12. ACM, 2018.

[PMM16] Kiril Panev, Evica Milchevski, and Sebastian Michel. Computing similar entity rankings via reverse engineering of top-k database queries. In *32nd IEEE International Conference on Data Engi-*

neering Workshops, ICDE Workshops 2016, Helsinki, Finland, May 16-20, 2016, pages 181–188. IEEE Computer Society, 2016.

[PMMP16] Kiril Panev, Sebastian Michel, Evica Milchevski, and Koninika Pal. Exploring databases via reverse engineering ranking queries with PALEO. *PVLDB*, 9(13):1525–1528, 2016.

[QWL+11] Jianbin Qin, Wei Wang, Yifei Lu, Chuan Xiao, and Xuemin Lin. Efficient exact edit similarity query processing with the asymmetric signature scheme. In Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 1033–1044. ACM, 2011.

[QYC12] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Diversifying top-k results. *CoRR*, abs/1208.0076, 2012.

[RBS10] Davood Rafiei, Krishna Bharat, and Anand Shukla. Diversifying web search results. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 781–790. ACM, 2010.

[RLB15] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. Matching HTML tables to dbpedia. In Rajendra Akerkar, Marios D. Dikaiakos, Achilleas Achilleos, and Tope Omitola, editors, *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, WIMS 2015, Larnaca, Cyprus, July 13-15, 2015*, pages 10:1–10:6. ACM, 2015.

[RPM16] Fabian Reinartz, Koninika Pal, and Sebastian Michel. Mining entity rankings. *Datenbank-Spektrum*, 16(1):27–38, 2016.

[SAM98] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven exploration of OLAP data cubes. In Hans-Jörg Schek, Fèlix Saltor, Isidro Ramos, and Gustavo Alonso, editors, *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings*, volume 1377 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 1998.

[San08] Evan Sandhaus. The new york times annotated corpus. LDC2008T19. DVD. Philadelphia: Linguistic Data Consortium, 2008.

[SC14] Sunita Sarawagi and Soumen Chakrabarti. Open-domain quantity queries on web tables: annotation, response, and consensus

models. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 711–720. ACM, 2014.

[SCC+01] Aya Soffer, David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, and Yoëlle S. Maarek. Static index pruning for information retrieval systems. In W. Bruce Croft, David J. Harper, Donald H. Kraft, and Justin Zobel, editors, *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*, pages 43–50. ACM, 2001.

[SH15] Markus Schedl and David Hauger. Tailoring music recommendations to users by considering diversity, mainstreaminess, and novelty. In Ricardo A. Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 947–950. ACM, 2015.

[SJPB08] Gleb Skobeltsyn, Flavio Junqueira, Vassilis Plachouras, and Ricardo A. Baeza-Yates. Resin: a combination of results caching and index pruning for high-performance web search engines. In Sung-Hyon Myaeng, Douglas W. Oard, Fabrizio Sebastiani, Tat-Seng Chua, and Mun-Kew Leong, editors, *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, pages 131–138. ACM, 2008.

[SK04] Sunita Sarawagi and Alok Kirpal. Efficient set joins on similarity predicates. In Gerhard Weikum, Arnd Christian König, and Stefan Deßloch, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 743–754. ACM, 2004.

[SLC10] Mingxuan Sun, Guy Lebanon, and Kevyn Collins-Thompson. Visualizing differences in web search algorithms using the expected weighted hoeffding distance. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 931–940. ACM, 2010.

[SP12] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *PVLDB*, 5(5):430–441, 2012.

[Spe94] C. Spearmsn. The american journal of psychology. *ACM Trans. Database Syst.*, 15(1):72–101, 1094.

[SPS+01] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alexander J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[SS00] Sunita Sarawagi and Gayatri Sathe. i^3: Intelligent, interactive investigaton of OLAP data cubes. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, page 589. ACM, 2000.

[SS01] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[SSWB00] Bernhard Schölkopf, Alexander J. Smola, Robert C. Williamson, and Peter L. Bartlett. New Support Vector Algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

[ST95] Abraham Silberschatz and Alexander Tuzhilin. On subjective measures of interestingness in knowledge discovery. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95), Montreal, Canada, August 20-21, 1995*, pages 275–281. AAAI Press, 1995.

[SvZ09] Frans Schalekamp and Anke van Zuylen. Rank aggregation: Together we're strong. In Irene Finocchi and John Hershberger, editors, *Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments, ALENEX 2009, New York, New York, USA, January 3, 2009*, pages 38–51. SIAM, 2009.

[SW49] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication.* Univ. of Illinois Press, 1949.

[TD01] David M. J. Tax and Robert P. W. Duin. Uniform object generation for optimizing one-class classifiers. *Journal of Machine Learning Research*, 2:155–173, 2001.

[THY+17] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. Extracting top-k insights from multi-dimensional data. In

Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1509–1524. ACM, 2017.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005.

[TTZ07] Yohannes Tsegay, Andrew Turpin, and Justin Zobel. Dynamic index pruning for effective caching. In Mário J. Silva, Alberto H. F. Laender, Ricardo A. Baeza-Yates, Deborah L. McGuinness, Bjørn Olstad, Øystein Haug Olsen, and André O. Falcão, editors, *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*, pages 987–990. ACM, 2007.

[TYSK10] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.*, 35(3):20:1–20:46, 2010.

[Uhl91] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[Vap95] Vladimir N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[VBMS96] T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors. Morgan Kaufmann, 1996.

[VCC99] K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. In *Proceedings of the International Joint Conference on AI*, pages 55–60, 1999.

[VG05] Anthony J Viera and J. Marshall Garrett. Understanding interobserver agreement: the kappa statistic. *Family medicine*, 37 5:360–3, 2005.

[WC03a] Gang Wu and Edward Y. Chang. Adaptive feature-space conformal transformation for imbalanced-data learning. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 816–823. AAAI Press, 2003.

[WC03b] Gang Wu and Edward Y. Chang. Class-boundary alignment for imbalanced dataset learning. In *In ICML 2003 Workshop on Learning from Imbalanced Data Sets*, pages 49–56, 2003.

[WKQ+08] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2008.

[WLF12] Jiannan Wang, Guoliang Li, and Jianhua Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 85–96. ACM, 2012.

[WLMJ14] Xiaoguang Wang, Xuan Liu, Stan Matwin, and Nathalie Japkowicz. Applying instance-weighted support vector machines to class imbalanced datasets. In Jimmy J. Lin, Jian Pei, Xiaohua Hu, Wo Chang, Raghunath Nambiar, Charu C. Aggarwal, Nick Cercone, Vasant G. Honavar, Jun Huan, Bamshad Mobasher, and Saumyadipta Pyne, editors, *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, pages 112–118. IEEE, 2014.

[WLWZ12] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Qili Zhu. Probase: a probabilistic taxonomy for text understanding. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 481–492. ACM, 2012.

[WMZ10] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.*, 28(4):20:1–20:38, 2010.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205. Morgan Kaufmann, 1998.

[WSSJ14] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[WWWZ12] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Qili Zhu. Understanding tables on the web. In Paolo Atzeni, David W.

Cheung, and Sudha Ram, editors, *Conceptual Modeling - 31st International Conference ER 2012, Florence, Italy, October 15-18, 2012. Proceedings*, volume 7532 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2012.

[WYY00] Wei Wang, Jiong Yang, and Philip S. Yu. Efficient mining of weighted association rules (WAR). In Raghu Ramakrishnan, Salvatore J. Stolfo, Roberto J. Bayardo, and Ismail Parsa, editors, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 270–274. ACM, 2000.

[XWL+11] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15:1–15:41, 2011.

[YDHJ07] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and Douglas Stott Parker Jr. Map-reduce-merge: simplified relational data processing on large clusters. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 1029–1040. ACM, 2007.

[YGCC12] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 97–108. ACM, 2012.

[ZGZG11] Peng Zhang, Byron J. Gao, Xingquan Zhu, and Li Guo. Enabling fast lazy learning for data streams. In Diane J. Cook, Jian Pei, Wei Wang, Osmar R. Zaïane, and Xindong Wu, editors, *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 932–941. IEEE Computer Society, 2011.

[Zha17] Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 8(6):921–957, 2017.

[ZM06] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 2006.

[ZSAR98] Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate similarity retrieval with m-trees. *VLDB J.*, 7(4):275–293, 1998.

List of Figures

List of Algorithms

List of Tables

Index

Acknowledgement

First of all, I would like to thank Sebastian for offering me this opportunity to work in such an excellent research facility. His continuous guidance, encouragement, and support made this academic journey a very enjoyable and memorable experience. The many valuable discussions we had helped me to grow as a researcher and blend myself comfortably with the working environment in Germany.

I am deeply grateful to all my colleagues, specially Evica, Kiril, and Manuel, for sharing their academic insights and valuable opinions in all the way throughout this journey and for making my stay enjoyable in Kaiserslautern by their cheerful presence.

Apart from this, a major thanks goes to Heike and Steffen, for helping me to adjust with the life in Germany by supporting me in several day-to-day and administrative issues from the very beginning.

I am very thankful to my friends in Saarbrücken and Kaiserslautern, who made the last few years a truly amazing, multi-colored experience. Life would have been hard without their friendship, warmth, and care. I would like to thank everyone of my old friends from India for their priceless friendships, shared experiences, and beautiful memories.

Many thanks goes to my family. In particular, I would like to thank Arijit—without him this whole experience would not be possible. And, last but not the least, I am grateful to my Ma, Baba, Alo, and Bua for their love and support and for staying by my side for all my whims, stupidities, and ventures.

Koninika Pal

DBIS Group
Campus E1 4, D5, MPII
Saarbrücken, Germany

Education

Sept. 2018 - present
Post Doctoral Research Fellow, *Department D5, Database and Information System, Max Planck Institut Informatik*, Saarbrücken, Germany.

Mar. 2014 - Sept. 2018
PhD in Computer Science, *TU Kaiserslautern (Jan. 2015 - Aug. 2018), Saarland University (Mar. 2014 - Dec. 2014),Germany*.
Thesis: Mining and Querying Ranked Entities.
Adviser: Prof. Dr. –Ing. Sebastian Michel

Apr. 2013 - Mar. 2014
Preparatory Phase in Graduate School of Computer Science, *Saarland University*, Saarbrücken, Germany.

Jul. 2010 - Jun. 2012
M.Tech in Information Technology, *National Institute of Technology (NIT), Durgapur, India*, CGPA: 9.48/10.
Thesis: Synchronization and QoS issues in Real Time Multimedia Communication in IP Network. Adviser: Dr. Animesh Dutta

Jul. 2005- Jun. 2009
B.Tech in Computer Science and Engineering Mechanical Engineering, *West Bengal University of Technology (WBUT), Kolkata, India*, CGPA: 8.53/10.
Thesis : ELIXIR COMPLETE REFERENCE TO CLINICAL DIAGNOSIS - A software designed on Database Management System

Research Experience

Mar. 2014 - Aug. 2018
TU Kaiserslautern and MMCI, Saarland University, *Germany*.
- Mining and maintaining semantically meaningful ranking
- Efficient similarity search over ranked lists

Jun. 2012 - Dec. 2012
Research Fellow (SPO), *Indian Institute of Technology*, Kharagpur, India.
- Building virtual sensor network
- Development of feasibility assessment model of adaptation in coal gasification technology using sensor network

Selected Attended Courses

Graduate Courses

Saarland University
- Information Retrieval and Data Mining
- Information Extraction and Semantic Web
- Optimization

NIT Durgapur
- Advance Algorithms
- Information Security and Risk Management

- Soft Computing

Undergraduate Courses

WBUT, Kolkata.

- Database and Management System (Theory & Practical)
- Data Structures and Algorithm (Theory & Practical)
- Artificial Intelligence (Theory & Practical)

Research Interests

- Exploring and Querying Knowledge Graph
- Mining Semantically Meaningful Rankings
- Entity Annotation for Web Tables
- Querying, Indexing and Performance study for Similarity Search on Big data

Publications

July, 2018 **Learning Interesting Attributes for Automated Data Categorization**, *Koninika Pal and Sebastian Michel*, Conference on Scientific and Statistical Database Management (SSDBM), Bolzano-Bozen, Italy.

May, 2017 **LSH-Based Probabilistic Pruning of Inverted Indices for Sets and Ranked Lists**, *Koninika Pal and Sebastian Michel*, 20th International workshop on the Web and Databases (WebDB) co-located with ACM SIGMOD 2017, Chicago, USA.

Sept, 2016 **Exploring Databases via Reverse Engineering Ranking Queries with PALEO**, *Kiril Panev, Sebastian Michel, Evica Milchevski, Koninika Pal*, PVLDB 9(13), India.

July, 2016 **Efficient Similarity Search across Top-k Lists under the Kendall's Tau Distance**, *Koninika Pal and Sebastian Michel*, Conference on Scientific and Statistical Database Management (SSDBM), Budapest, Hungary.

March, 2016 **A Data Mining Approach to Choosing Categorical Attributes for Ranked Lists**, *Koninika Pal and Sebastian Michel*, 19th International Conference on Extending Database Technology (EDBT, Poster), Bordeaux, France.

2016 **Mining Entity Rankings**, *Fabian Reinartz, Koninika Pal and Sebastian Michel*, Datenbankspektrum, Vol. 16(1), pp 27-38, Germany.

June, 2014 **An LSH Index for Computing Kendall's Tau over Top-k Lists**, *Koninika Pal and Sebastian Michel*, 17th International workshop on the Web and Databases (WebDB) co-located with ACM SIGMOD 2014, Utah, USA.

2012 **Multipoint Multimedia Synchronization: A Petri Net Based Approach**, *Animesh Dutta and Koninika Pal*, INFOCOMP Journal of Computer Science vol. 11(2).

Sept, 2011 **A Petri Nets Based Model for Multipoint Synchronization in Multimedia conferencing**, *Koninika Pal, Prajna Devi Upadhyay and Animesh Dutta*, SUComS 2011, Hualien, Taiwan.

Jan, 2011 **A Petri Nets Based Model for Multipoint Synchronization in Audio Conferencing**, *Koninika Pal, Prajna Devi Upadhyay and Animesh Dutta*, COMSNET 2011, Bangalore, India.

Teaching Experiences

SS-2015, SS-2017 — **Teaching Assistant**, *Course: Information retrieval and Data Mining*. TU Kaiserslautern, Germany

WS 2016/17, WS 2017/18 — **Organizer**, *Seminar: Recent Developments in Databases and Information Systems*. TU Kaiserslautern, Germany

WS 2015/16 — **Organizer**, *Project: Information System (Building Meta Search Engine for Document Retrieval)*, TU Kaiserslautern, Germany.

Jan-July, 2016 — **Supervisor**, *Masters Thesis: Mining Feasible Description for a Set of Entities*, Arghavan Hosseinzadeh da Silva, TU Kaiserslautern, Germany.

August, 2017 - April, 2018 — **Supervisor**, *Masters Thesis topic: Annotating Web Tables by exploiting relations among table columns*, Michael Hohenstein, TU Kaiserslautern, Germany.

Nov, 2017 - May, 2018 — **Supervisor**, *Guided Research topic: Making sense of the distribution of ranking criteria*, Gajendra Doniparthi, TU Kaiserslautern, Germany.

Technical Skills

Programming — JAVA, SQL, C, C++, PYTHON

Operating Systems — Linux (Ubuntu), MacOS, Windows

Other Activities

- Research Visit and talk in Maya Ramanath's group in Indian Institute of Technology, Delhi, India, September, 2016
- Attended 1st ACM Europe Summer School on Data Science, Athens, Greece, July, 2017

Personal Information

Nationality — Indian

Languages — Bengali (Native), English(C1), German(A2)