Data De-duplication: A Review

Gianni Costa, Alfredo Cuzzocrea, Giuseppe Manco, and Riccardo Ortale

Abstract. The wide exploitation of new techniques and systems for generating, collecting and storing data has made available growing volumes of information. Large quantities of such information are stored as free texts. The lack of explicit structure in free text is a major issue in the categorization of such kind of data for more effective and efficient information retrieval, search and filtering. The abundance of structured data is problematic too. Several databases are available, that contain data of the same type. Unfortunately, they often conform to different schemas, which avoids the unified management of even structured information. The *Entity Resolution* process plays a fundamental role in the context of information integration and management, aimed to infer a uniform and common structure from various large-scale data collections, with which to suitably organize, match and consolidate the information of the individual repositories into one data set. *De-duplication* is a key step of the Entity Resolution process, whose goal is discovering duplicates within the integrated data, i.e., different tuples that, as a matter of facts, refer to the same real-world entity. This attenuates the redundancy of the integrated data and, also, enables more effective information handling and knowledge extraction through a unified access to reconciled and de-duplicated data. Duplicate detection is an active research area that benefits from contributions from diverse research fields, such as, machine learning, data mining and knowledge discovery, databases as well as information retrieval and extraction. This chapter presents an overview of research on data de-duplication, with the goal of providing a general understanding and

Gianni Costa · Alfredo Cuzzocrea · Giuseppe Manco · Riccardo Ortale
ICAR-CNR, Via P. Bucci, 41C, 87036 Rende (CS) - Italy
e-mail: `{costa,cuzzocrea,manco,ortale}@icar.cnr.it`

useful references to fundamental concepts concerning the recognition of similarities in very large data collections. For this purpose, a variety of state-of-the-art approaches to de-duplication is reviewed. The discussion of the state-of-the-art conforms to a taxonomy that, at the highest level, divides the existing approaches into two broad classes, i.e., unsupervised and supervised approaches. Both classes are further divided into sub-classes according to the common peculiarities of the involved approaches. The strengths and weaknesses of each group of approaches are presented. Meaningful research developments to further advance the current state-of-the-art are covered as well.

1 Introduction

Recognizing similarities in large collections of data is a major issue in the context of information integration. The wide exploitation of new techniques and systems for generating, collecting and storing data has made available very large collections of data, such as personal demographic data, bibliographic information, phone and mailing lists. Often, the integration of such data is a problematic process, that involves dealing with two major issues, namely *structural* and *syntactic* heterogeneity.

Structural heterogeneity is essentially due to the lack of a common (explicit or latent) structure in the available data, that avoids a proper organization of such data for a more effective and efficient information retrieval, search and filtering. Structural heterogeneity depends on the nature of the available data. In the case of structured data, the individual repositories explicitly exhibit their own schemas and, thus, dealing with the arising structural heterogeneity involves finding a common schema for the representation of the integrated data. Instead, when the available data collections are unstructured, such as in the case of free text, it is likely that they are characterized by a latent segmentation into specific semantical entities (e.g., personal demographic information typically comprises names, addresses, zip codes and place names, which indicate a convenient organization for the these kind of data). Such a segmentation is not a-priori known and, hence, handling with structural heterogeneity of textual data means finding an explicit common schema from the various latent ones (if any). In principle, such a common schema would allow to fit the integrated textual data into some field structure, so that to exploit the mature relational technology for more effective information management. Structural heterogeneity is addressed through the exploitation of suitable methods and techniques that reconcile distinct collections of (structured or unstructured) data, by integrating them into one data set with a uniform schema.

However, once reconciled, the integrated data can be affected by syntactic heterogeneity. This is a fundamental issue in the context of information integration systems, that consists in discovering duplicates within the integrated data, i.e., syntactically different records that, as a matter of facts, refer to the same real-world entity. Duplicate detection is necessary to avoid data redundancy as well as inaccuracies in query processing and knowledge extraction [13]. A typical example is the reconciliation of demographic data sources into a data warehousing environment. Consider, e.g., a banking scenario, where the main interest is to rank the credit risk of a customer, by looking at the past insolvency history. In this setting, useful information about payments may come from different sources, each of which likely conforming to a different encoding of the data, so that names and addresses may be stored in rather different formats. The reconciliation of the various data sources is a first step towards the design of a decision support system. However, it is the de-duplication of the reconciled data that allows to correctly analyze the attitude of insolvency of the individual customers, thus enabling an effective decision making.

More generally, besides the post-processing of reconciled data, de-duplication techniques are also particularly useful in all those applicative settings where large collections of data are available. In such domains, such techniques can be exploited in a preliminary exploratory phase, for the purpose of reducing the number of duplicated data, which ultimately improves the quality of the underlying data.

Four challenging requirements of duplicate detection are: (*i*) the capability to handle huge volumes of data; (*ii*) efficiency; (*iii*) scalability; (*iv*) the availability of incremental algorithms.

In particular, the requirement to process large bodies of data generally imposes severe restrictions on the design of data structures and algorithms for de-duplication. Such restrictions are necessary to ultimately limit both the computational complexity of the de-duplication scheme (a required time, that is quadratic in the size of the underlying database, is prohibitively high) and its I/O operations on disk (several random accesses to secondary storage imply continuous paging activities).

Efficiency and scalability issues do play a predominant role in many applicative contexts, where large data volumes are involved, especially when the object-identification task is part of an interactive application, calling for short response times. For instance, the typical volume of data collected on a daily basis in a banking context amounts on average to $500,000$ instances, representing credit transactions performed by customers throughout the various agencies. In such a case, the naive solution of comparing such instances in a pairwise manner, according to some given similarity measure, is infeasible. As an example, for a set of $30,000,000$ tuples (i.e., data collected in a 2 months-monitoring), the naive strategy would require $O(10^{14})$ tuple comparisons, which is clearly prohibitive.

Also, the detection of duplicates should preferably be performed in an incremental manner, so that to properly account for the possibly streaming nature of the data. In such cases, the available data collection is incrementally augmented through the progressive integration of newly arrived data, whose prior de-duplication raises a stringent online requirement, i.e., that the redundancy of such data is promptly recognized. Practically speaking, the cost of incremental de-duplication should be (almost) independent of the size of the available data.

This chapter surveys seminal research in the field of duplicate detection and discusses consolidated results in the light of the aforementioned requirements. The discussion proceeds as follows. Section 2 introduces the process of duplicate detection and identifies two categories of approaches, namely, supervised and unsupervised techniques. Section 3 deals with the supervised approaches. Section 4 is devoted to the unsupervised approaches. In both sections 3 and 4, the strengths and the weaknesses of each encountered family of approaches are discussed. Finally, Section 5 concludes by highlighting directions of further research, that can advance the current state-of-the-art in duplicate detection.

2 Problem Description

Duplicate detection is a step of a more complex process, referred to as *Entity Resolution* [11, 13, 26, 41, 47, 73, 98], that plays a fundamental role in the context of information integration and management. The typical scenario in the design of information systems is the availability of multiple data repositories, with different schemas and assumptions on the underlying canonical data representation. Schema differences imply a segmentation of data tuples[1] into sequences of strings, that correspond to specific semantical entities. However, such a segmentation is not known in advance and this is a challenging issue for duplicate detection. Moreover, the adoption of various canonical data representations (such as the presence of distinct data separators and/or various forms of abbreviations) coupled with erroneous data-entry, misspelled strings, transposition oversights and inconsistent data collection further exacerbates the foresaid difficulties behind the recognition of duplicates. The goal of Entity Resolution is to suitably reconcile, match and consolidate the information within the different repositories [14], so that all data exhibits a uniform representation and duplicates are properly identified. The process of Entity Resolution consists of the following three steps.

- *Schema reconciliation*, which consists in the identification of a common representation for the information in the available data [1, 18, 21, 66, 71, 77].
- *Data reconciliation*, that is the act of discovering synonymies in the data, i.e. apparently different records that, as a matter of fact, refer to the same real-world entity.
- *Identity definition*, aimed to find groups of duplicate tuples and to extract one representative tuple for each discovered group. Representative tuples allow better information processing as well as a meaningful compression of the size of the original data.

The focus of this chapter in on the techniques for the detection of duplicated data, typically employed in the second step of the Entity Resolution process. Duplicate detection has given rise to a large body of works in several research communities, where it is referred to with as many umbrella names, such as, e.g., *Merge/Purge* [53], *Record Linkage* [39, 97], *De-duplication* [85], *Entity-Name Matching* [30], *Object Identification* [79].

[1] The term *tuple* is abstractedly used throughout the chapter to denote a reconciled fragment of data, that can be either a textual sequence of strings or a structured record.

In most of these approaches, a major issue is represented by the definition of a method for comparing tuples, that typically consist of many components. Recognizing similar tuples involves matching their constituting components. These can represent both numbers and strings. Depending on the structured or unstructured nature of the underlying data, such components can be either values of the fields of some database schema or tokens from textual data. In both cases, approaches to data de-duplication require dealing with mismatches between tuple components. While research on the identification of mismatches between numbers is not yet mature, a variety of schemes have been developed for dealing with various kinds of heterogeneity and mismatches between strings across different information sources [58].

A categorization of the most commonly adopted schemes for matching string components in the context of duplicate detection is provided in [58]. Such a categorization includes two major classes of string (dis)similarity functions [50], i.e., *character-based similarity metrics*, that are meant for dealing with differences in the individual characters of string components, and *token-based similarity metrics*, that instead aim to capture a degree of similarity between two tuples, even if their components are rearranged.

A detailed analysis of these metrics is beyond the scope of this chapter. For our purposes, it suffices to know that the availability of such schemes for matching individual component allows the design of suitable approaches to the detection of duplicates in the domains of both structured and textual data [15, 29, 75, 76, 85], in which individual tuples consist of multiple components. These approaches can be divided into two broad and widespread families, i.e., supervised and unsupervised techniques, which are respectively covered in section 3 and section 4.

3 Supervised Approaches to De-duplication

The common idea behind such category of approaches is to learn from the training data suitable models for tuple matching [6, 81, 92, 36]. This involves learning probabilistic models [39, 82] or deterministic classifiers [15, 30] characterizing pairs of duplicates from training data, consisting of known duplicates. Such methods assume that training data contain the wide variety of possible errors in practice. However, such a comprehensive collection of training data very rarely exists in practice. Partially, this issue was addressed by approaches based on active learning [85, 91]. These require an interactive manual guidance. In many scenarios, it is not plain to obtain satisfying training data or interactive user guidance.

For what specifically regards supervised approaches to de-duplication, we argue that, being state-of-the-art proposals of the literature very heterogenous among them, the most convenient classification to be proposed here is the one based on the *kind of data* to be de-duplicated, and can be reasonably devised as follows: (*i*) *supervised approaches for de-duplicating relational data*; (*ii*) *supervised approaches for de-duplicating multidimensional data*; (*iii*) *supervised approaches for de-duplicating Data-Mining data/results*; (*iv*) *supervised approaches for de-duplicating linked and XML data*; (*v*) *supervised approaches for de-duplicating* *streaming data*. The remaining part of this Section is organized in sub-sections that strictly follow the proposed classification for supervised approaches to de-duplication.

3.1 Relational Data De-duplication Approaches

In the context of relational data, [75] proposes an efficient algorithm for recognizing *clusters of approximatively duplicate records* in large databases. The main goal of the proposed algorithm consists in overcoming limitations exposed by a previous relevant approach, the *Smith-Waterman algorithm* [89], which has been early proposed for identifying common molecular subsequences in the context of molecular biology research, by means of three innovative steps. First, an optimized version of the Smith-Waterman algorithm is introduced in order to compute minimum edit distances among candidate duplicate records according to a *domain-independent approach*. Second, a meaningful union algorithm for keeping track of duplicate clusters as long as new duplicates are discovered is exploited. Third, a novel priority-queue-based method is used to finally retrieve clusters of duplicate records depending on the size and the homogeneity of the underlying database. Experimental results demonstrate the benefits of algorithm [75] over the state-of-the-art Smith-Waterman algorithm.

[35] moves the attention on the problem of improving *spatial join algorithms over spatial databases* via detecting duplicates that may occur in the set of candidate spatial database objets involved by (spatial join) queries. The most remarkable contribution of this research consists of a significant improvement of performance of two state-of-the-art spatial join algorithms, namely *PBMS* [83] and S^3J [67], as confirmed by experiential results shown in [35]. [35] clearly demonstrates how duplicate record/object detection not only is useful for Data Mining and Data Warehousing, but even for Database query processing issues.

Blocking techniques [61] represent a traditional indexing approach for reducing the number of comparisons due to data de-duplication, at the cost of potentially missing some true matches. These techniques typically divide the target database into *blocks* and compare only the records that fall into the same block. One traditional method is to scan the database and compute a value of an appropriate *hash function* for each record. The value of the hash function defines the block to which this record is assigned. The limit of this approach is that conventional hashing techniques cannot be used for obtaining approximate duplicate detections, since the hash value of two similar records could not be the same (due to unexpected collisions).

[85] focuses the attention on the presence of duplicates in *data integration systems*, hence it elects de-duplication as one of the most important research challenge to be faced-off in such systems. Indeed, integrating data from multiple and heterogeneous sources easily exposes to the presence of duplicates, both data and concept duplicates. Starting from limitations of actual approaches, which are mainly

hand-coded in nature, [85] proposes an innovative *learning-based de-duplication method* whose main idea consists in interactively discovering critical training pairs, i.e. those training pairs that may effectively provide a benefit for the ongoing de-duplication process, via so-called *active learning* [31]. This method is then embedded into the core layer of a complete interactive de-duplication system, called `ALIAS`, for which reference architecture and main functionalities are provided. Overall, [85] indeed proposes a sort of *learning-based classifier*, whose performance is assessed via a comprehensive set of experiments on both synthetic and real-life data sets.

Christen [16] provides us with an extremely useful research experience on the performance and scalability of so-called blocking techniques [61] for data de-duplication. This study first depicts a meaningful "high-level" view about the general data de-duplication process, and then a very comprehensive experimental analysis and evaluation of performance issues as well as scalability aspects of blocking techniques, which puts in emphasis benefits and limitations of state-of-the-art approaches, mostly focusing on main-memory management and time complexity results. As a secondary, yet useful, result of [16] the author concludes that blocking techniques are very sensitive to the ranging of parameter values, and, since finding the optimal setting for these parameters depends on the quality and the specific characteristics of the databases to be de-duplicated, it is not easy to apply these techniques in real-life database settings, where the ranging above can become arbitrary and completely-impredicative at all. The latter one is another significant lesson to be learned with respect to data de-duplication research principles.

Christen again proposes the *Febrl* (Freely Extensible Biomedical Record Linkage) system in [17]. *Febrl* is an open source tool, provided with a nice graphical user interface, which embeds a relevant number of state-of-the-art data de-duplication approaches in order to improve data cleaning and standardization in the domain of health databases of Australia. The most distinctive characteristic of *Febrl* relies in the fact that it is prone to house and integrate any novel arbitrary data de-duplication technique one would like to embed in the system, hence providing interesting extensible and cross-analysis facilities that allow researchers to boost the finding capabilities in this so-interesting scientific area. In [17], Christen provides us with a very detailed description of architecture and main functionalities of *Febrl*, plus discussion on data de-duplication tasks end-users are allowed to define and run in this open source environment.

Finally, [51] focuses the attention on studying the *quality* of de-duplication results due to *clustering-based approaches*, in order to discover limitations and potentialities of this family of data de-duplication methods. The analysis is conducted within the context of the *Stringer* system, a modular architecture that provides a reliable evaluation framework for stressing the effectiveness-on-arbitrary-domains and the scalability of clustering-based data de-duplication approaches. The result of this study makes us aware about some surprising evidences: some clustering algorithms that have never been considered for duplicate detection problems expose good performance as regards both accuracy and scalability of the data de-duplication phase. As a further result, in [51] authors conclude that it is not possible to obtain perfect duplicate clusters by means of actual methods available in literature, so that *quantitative approaches* (e.g., [52]) oriented to find accurate confidence scores for each duplicate cluster detected are mandatory.

3.2 Multidimensional Data De-duplication Approaches

In the context of Data Warehousing systems, [27] proposes an efficient *data reconciliation approach*, whose main benefit consists in introducing an *approximate matching method* that incorporates *Machine Learning* [74] tools and statistical techniques with the goal of sensitively reducing the spatio-temporal complexity due to the very large number of comparisons to be performed in order to detect matches across different data sources. The approach in [27], mainly proposed for data reconciliation purposes (which, in some sense, is more general than data de-duplication issues), can be straightforwardly adapted for data de-duplication purposes, yet preserving similar performance.

In [2], authors investigate the problem of detecting duplicate records in the relevant application scenario represented by Data Warehousing systems. To this end, authors propose and experimentally assess `DELPHI`, a duplicate detection algorithm tailored to the specific target of *dimensional tables* one can find in the data layer of Data Warehousing systems. `DELPHI` takes advantages from the semantics expressed by *dimensional hierarchies* available in such systems in order to improve the quality and the effectiveness of state-of-the-art approaches for duplicate record detection that, as authors correctly state, still expose a high number of false positives over conventional database tables.

[69] proposes a nice theoretical work focusing on general aspects of data cleaning for Data Warehousing. The idea carried out by [69] consists in a novel *knowledge-based approach* that exploits the degree of similarity of nearby records of large databases to detect duplicates within such databases. In more details, [69] trades-off the *recall* of the duplicate detection phase, i.e. the sensitivity of the target method in accepting as duplicates those records exposing a low degree of similarity with other records, and the *precision* of the duplicate detection phase, i.e. the sensitivity of the target method in accepting as duplicates only records having an high degree of similarity with other records. This conveys in the so-called *recall-precision dilemma* [69]. Authors propose a solution to this dilemma by means of a theoretical framework that makes use of *transitive-closure tools* over suitable graphs modeling the uncertainty of similarities among candidate records, and advocates for approximate solutions.

Finally, in [30] authors address the particular context represented by *large high-dimensional data sets*, and propose the innovative tasks of "*entity-name matching*" and "*entity-name clustering*", which, as authors claim, play critical roles in data cleaning over high-dimensional data [30]. In more detail, "*entity-name matching*" deals with the problem of taking two lists of entity names from two different sources and determining which pairs of names are co-referent (i.e., refer to the same real-world entity). The term "*entity-name clustering*" instead refers to the task of taking a single list of entity names and assigning entity names to clusters such that all names in a cluster are co-referent. The major benefit due to [30] consists in proposing an *adaptive* approach, meaning that accuracy of the duplicate detection phase can be progressively improved by training.

3.3 Data-Mining Data/Results De-duplication Approaches

In the context of Data Mining, Winkler [99] studies the conceptual/theoretical overlap between duplicate detection and *Bayesian networks*, which are well-understood tools of Machine Learning research [74]. In more detail, Winkler finds five special conditions such that the *Expectation Maximization* (EM) algorithm [34] can be used for parameter estimation within the core layer of Bayesian networks, in order to gain a significant speed-up during the duplicate detection phase performed by these networks. Hence, main results from [99] are of theoretical nature, and have been further exploited for subsequent research in the duplicate detection context. On the other hand, in [99] the author experimentally shows the performance gain due to applying the EM algorithm for making Bayesian networks faster on duplicate detection situations, and he also proves that this approach performs better than traditional approaches that are, generally, based on iterative refinement methods.

In [88], authors address the record de-duplication problem in the context of conventional *Knowledge Discovery in Databases* (KDD) [38] processes from a completely-novel perspective that, indeed, turns to be innovative with respect to previous research experiences. Here, the main idea consists in making use of an innovative *multi-relational approach* able of performing simultaneous inference for all the pairs of candidate duplicate records, while allowing the model information to propagate from one candidate match to another via exploiting the set of (database) attributes these candidates share in common. Based on this theoretical model for representing and processing candidate duplicate records simultaneously, authors design the guidelines of a general framework for supporting the so-called *collective inference* of possible duplicates. Analytical and theoretical results presented in [88] are fully-supported by a comprehensive campaign of experiments that truly demonstrate the benefits of the multi-relational de-duplication approach over state-of-the-art conventional ones on both syntectic and real-life data sets.

Finally, [44] deals with the problem of making data de-duplication methods *automatic*, in order to provide a more reliable and effective support to Data Mining processes, hence avoiding tedious manual clerical reviews needed to (manually) check possible data links that may still exist. It should be noted that the situation above would be not even possible in some real-life critical application scenarios where real-time linkage of streaming data is required (e.g., credit card transactions, public health survey systems, and so forth). The main proposal from authors in [44] consists in a *decision-tree-based approach* that embeds some state-of-the-art *compression methods* aimed at improving the efficiency of the data de-duplication phase. Combining efficient previous methods makes the approach proposed in [44] particularly suitable to support record de-duplication over very large data sets, perhaps via invoking well-understood *parallel processing* paradigms.

3.4 Linked and XML Data De-duplication Approaches

The *identity uncertainty* problem related to the context of citing research papers, which fall in the context of linked data, is investigated in [84], as a practical data/object de-duplication issue arising in real-life information systems. This challenge is becoming relevant for a large spectrum of modern application scenarios, as actually enormous, massive amounts of bibliographic knowledge is made available to open research communities via citation tools and more-conventional digital libraries via the Web. Hence, multiple observations may easily correspond to the same (bibliographic) object, thus introducing severe flaws during the fruition of bibliographic knowledge. Starting from these motivations, authors in [84] attack uncertainty of identities via innovative *probabilistic models* over mappings defined between terms and objects, and infer (probabilistic) matches via well-understood *Markov-Chain Monte Carlo* [42] solving paradigms on top of such models. One of the significant and singular particularities of the approach [84] is represented by some specific optimizations introduced in the classical Markov-Chain Monte Carlo solving method in order to generate efficient object candidates when the target domain contains a large number of object duplicates for a certain term. Experimental results shown in [84] demonstrate the effectiveness of the proposed identity uncertainty management approach over the bibliographic data set underlying the well-known Web citation system *Citeseer* [68].

Correlation clustering [8] is an elegant clustering method that was originally proposed for clustering graphs with binary edge labels modeling the existence of correlation or un-correlation between the connected nodes. As highlighted in previous studies, correlation clustering can be adapted as a solving method for data de-duplication problems. In fact, (*i*) nodes can store candidate duplicate data/objects, (*ii*) labels can be assigned to edges on the basis of user-defined similarity scores between pairs of nodes, and (*iii*) unconstrained clustering algorithms can be used to cluster duplicate data/objects, based on a given threshold over similarity scores.

Detecting duplicates in XML data, which are relevant for modern Web applications and systems, is first investigated in [95]. [95] particularly considers the interesting scenario of detecting duplicates for all the kinds of parent/child relationships that can occur in an XML data set. Previous studies have focused on the problem of detecting duplicates for $1 : n$ parent/child (XML) relationships only. The most relevant contribution of [95] consists in the fact that an innovative *comparison order* of pairwise classification is introduced, being this order able of reducing the number of (re)classifications of the same pair of candidate duplicate XML objects. Two different algorithms that efficiently exploit this novel ordering solution are presented, and experimentally tested on real-life movie data sets.

[96, 55] instead introduces the *duplicate detection problem for graph data*. Similarly to the case of XML data [95], here authors make use of relationships among objects (of the underlying database) in order to build a suitable graph modeling such relationships, and then study how to improve the effectiveness and the efficiency of traditional duplicate detection approaches over this graph, by reducing spatial and time complexity. Here, the main idea consists of an *hybrid approach* that encompasses an initialization phase and an iterative phase, both aimed at gaining performance over traditional solutions. Furthermore, the proposed framework argues to achieve high scalability over large amount of data thanks to a proper RDBMS layer, which provides support for some specific elementary routines of the duplicate-detection-in-graphs phase by means of very efficient SQL statements implemented in forms of well-understood *stored procedures*. In [96, 55], a wide and pertinent experimental assessment of the proposed RDBMS-supported framework on both synthetic and real-life data sets clearly demonstrates the effectiveness and the efficiency of the framework, even in comparison with traditional data de-duplication approaches.

Finally, [70] moves the attention on *duplicate detection over large XML documents*, as a further extension of previous work [96, 55]. The proposed method is based on the well-known cryptographic hashing algorithm *Message Digest algorithm 5* (MD5) by Rivest *MD5 algorithm* [12], an efficient algorithm that takes as input messages of arbitrary length and returns as output message digests of fixed length, and it encompasses three different modules: (*i*) the *selector*, which retrieves candidate duplicate objects from the input XML document and a fixed set of candidate definitions; (*ii*) the *pre-processor*, which pre-processes candidate duplicate objects in order to code them into 512-bit padded messages; (*iii*) the *duplicate identifier*, which finally selects the duplicate objects based on the MD5 algorithm running on the padded messages generated by the pre-processor module.

3.5 Streaming Data De-duplication Approaches

Finally, in the context of streaming data, [100] addresses the relevant research challenge represented by *detecting duplicates in streaming data*, which has received considerable attention from the Database and Data Mining research communities (e.g., [87]). Indeed, in the context of data stream processing, renouncing to the uniqueness assumption on observed data items from an input stream is very demanding, as almost all the actual *aggregation approaches* over streaming data available in literature would need significant revision at both the theoretical and the implementation-wise level. Based on this key observation, authors propose a family of techniques able of computing *duplicate-insensitive order statistics* over data streams, with provable error guarantees. The proposed techniques are proven to be space- and time-efficient and suitable to support on-line computation of very high-speed data streams. Authors complete their nice analytical and theoretical contributions by means of a comprehensive set of experiments on both syntectic and real-life data stream sets, which further confirm the effectiveness and the efficiency of the proposed techniques.

4 Unsupervised Approaches to De-duplication

A major disadvantage of supervised approaches to duplicate detection is the requirement for appropriate amounts of labeled training data, which involves a considerable human effort in labeling pairs of training data as either duplicates or non-duplicates. This task becomes especially challenging when it comes to provide examples of ambiguous cases (e.g., apparently duplicate tuples that are really non duplicates and viceversa), from which to learn more effective models, that are capable to sharply discriminate duplicates from non-duplicates [58]. In practice, supervised approaches assume the availability of training data explaining the wide variety of possible errors for each targeted entity. However, such comprehensive collections of training data very rarely exist. Partially, this issue was addressed by resorting to active learning [85, 91]. Unfortunately, this still requires an interactive manual guidance. As a matter of fact, in many practical applications of supervised duplicate detection, it is not plain to obtain satisfying training data or interactive user guidance.

To avoid the limitations of supervised duplicate detection, a large body of unsupervised approaches to data de-duplication has been proposed in the literature. Such approaches essentially define suitable techniques for grouping duplicate tuples, so that to minimize two types of incorrect matchings: false-positives (i.e., tuples recognized as similar, that actually do not correspond to the same entity) and false-negatives (i.e., tuples corresponding to the same entity, that are not recognized as similar). The pursuit of such objectives has largely prompted the design of unsupervised classification methods, mostly based on clustering or nearest-neighbor classification. Therein, in order to meet the earlier requirements on effectiveness, efficiency and scalability, various categories of schemes for approaching de-duplication in terms of unsupervised classification have been developed. We focus on three major categories of unsupervised de-duplication schemes, which are discussed separately in the rest of this section. Precisely, subsection 4.1 covers the exploitation of consolidated clustering schemes for duplicate detection. Subsection 4.2 is devoted to de-duplication via (dis)similarity-search in metric spaces. Finally, subsection 4.3 deals with duplicate-detection through locality-sensitive hashing.

4.1 De-duplication Based on Clustering

Clustering methods [45, 59, 60] have been exploited for the de-duplication purpose to divide a set of tuples into various clusters. The individual clusters refer to corresponding real-world entities and meet the following two requirements: *homogeneity*, i.e. pairs of tuples within a same cluster are highly similar and, hence, expected to be duplicates, and *neatly separation*, i.e. pairs of tuples within distinct clusters are very dissimilar and, therefore, deemed to refer to distinct real-world entities.

A variety of clustering methods can be exploited in the context of duplicate detection. A very effective approach would be using a hierarchical clustering method[2], equipped with an accurate component-wise similarity metric, such as edit distance, affine gap distance, smith-waterman distance and Jaro distance (see [58] for a detailed survey) to match tuple tokens. Unfortunately, the quadratic complexity of hierarchical clustering in the number of available tuples, combined with the high computational cost of the schemes for matching tuple components (that becomes quadratic w.r.t. the length of tokens in the case of edit distance), would penalize the efficiency and scalability of the resulting de-duplication process, up to the point of making the latter impractical in the great majority of applicative domains, where even a small amount of data is available.

Apart from hierarchical methods, several consolidated clustering algorithms [37, 40, 48, 49, 53] are at the heart of various de-duplication techniques. However, although generally effective, these techniques do not generally guarantee an adequate level of scalability. As a matter of fact, these approaches would not work adequately in a scenario, where far too many clusters are expected to be found, as it does happen in a typical de-duplication scenario, where the actual number of clusters of duplicate tuples can be of the same order as the size of the database. The only suitable approaches appear to be the ones in [23, 72].

Precisely [72] avoids costly pairwise comparisons by grouping objects in *canopies*, i.e., subsets containing objects suspected to be similar according to some cheap (i.e. computationally inexpensive) similarity function and, then, computing actual pairwise similarities only within the discovered canopies. Since in a typical duplicate detection scenario there are several canopies, and an object is shared in a very few number of canopies, the main issue of the approach is the creation of canopies.

In [23], an efficient two-phase approach is proposed: first determine the nearest neighbors of every tuple in the database and, then, partition the original collection into groups of duplicates. The efficiency of the algorithm strictly relies on the nearest neighbors computation phase, where the availability of any disk-based index (i.e., inverted index associated with edit or fuzzy similarity functions) is assumed. Efficiency comes from the lookup order in which the input tuples are scanned, in order to retrieve nearest neighbors. The order corresponds to a breadth first traversal of a tree, where the children of any node are its nearest neighbors. The benefit consists in accessing, for consecutive tuples, the same portion of the index, thus improving the buffer hit ratio.

Despite their strengths, the approaches in [23, 72] are not meant for incremental de-duplication.

[2] Hierarchical clustering algorithms are well known in the literature for producing top quality results [60].

4.2 De-duplication Based on (dis)Similarity-Search in Metric Spaces

De-duplication can also be performed by grouping duplicates through neighbor-driven clustering. Given a collection of tuples equipped with a suitable (dis)similarity metric, the idea is that the cluster membership of a tuple should be established by looking at the clusters to which other similar tuples belong.

Neighbor-driven de-duplication consists of three basic steps. Initially, the pair-wise distance between tuples is computed. Then, a list of *neighbors* is retrieved for each query tuple, that is, for each tuple to be de-duplicated. A tuple is essentially a neighbor of the query tuple, when the former is similar to the latter according to the adopted similarity metric. Ultimately, the cluster membership for the query tuple is determined through a *voting* procedure, in which the retrieved neighbors, that are likely duplicates of the query tuple, vote for the cluster to which the latter should be assigned. The most basic voting scheme is the *majority* one, in which the query tuple is assigned to the cluster that is most common among its neighbors.

Similarity-search [24, 56] plays a major role in neighbor-driven de-duplication, since it allows the identification of all neighbors of a query tuple. However, the high dimensionality [94] of the space in which the search is typically performed is a major weakness of neighbor-driven de-duplication. Moreover, similarity-search requires setting an appropriate upper bound (or, also, a threshold) to the maximum distance from the query tuple, that actually identifies the neighborhood of the latter. Computing a distance threshold is problematic, since, as it is pointed out in [23], one absolute global distance threshold does not guarantee an effective retrieval of neighbors. This has undesirable effects. Indeed, the identification of too few neighbors may make duplicate clustering susceptible to overfitting. Instead, too many neighbors may lead to noisy duplicate detection, since far dissimilar tuples may be involved in the voting procedure. According to the results in [23], the distance threshold should be actually considered as a local property of the individual group of duplicates [23]. Therefore, each real-world entity in the data should require the computation of a suitable distance threshold, that differs from the thresholds associated with the other entities in the same data. Unfortunately, the tuning of several distance thresholds would make the resulting de-duplication process impractical. Additionally, the basic similarity-search for neighboring duplicates involves computing similarities between all pairs of tuples in the underlying data, thus being computationally quadratic in the overall number of available tuples. This exceedingly penalizes both the efficiency and the scalability of de-duplication.

In order to speed up the basic approach to neighbor-driven de-duplication, various refinements have been proposed [2, 22, 46], that exploit efficient indexing schemes. Unfortunately, these refinements are not specifically designed to approach neighbor-driven de-duplication from an incremental clustering perspective. Therein, neighbor-driven clustering would in principle benefit from an indexing scheme, that supports the execution of similarity queries and can even be incrementally updated with new tuples. Nonetheless, the syntactic heterogeneity of the tuples at hand is likely to heavily increase the size of the index, which would ultimately degrade the performance of the overall neighbor-driven de-duplication process. To further elaborate on this point, the exploitation of an indexing scheme for neighbor-driven de-duplication was empirically investigated in [32]. In particular, this study focused on the *M-Tree* index [25], a well-known, state-of-the-art index/storage structure, which looks like a *n*-ary tree.

The M-Tree allows to index and organize tuples, provided that a suitable distance metric *dist* is defined for pairwise tuple comparison. Precisely, tuples are arranged into a balanced tree structure, in which each node has a fixed size (related to the size of a page to be stored on disk). The individual entries of a non-leaf node store *routing* objects, i.e., summaries of the information about the contents of the subtrees rooted at the children of that node. In turn, each routing object O_r is associated with two further elements: a *pointer* referencing the root of a sub-tree $T(O_r)$ (the so-called *covering tree* of O_r) and a *covering radius* $r(O_r)$, which guarantees that all objects in $T(O_r)$ are within the distance $r(O_r)$ from O_r. The search for the neighbors of a query tuple t can be efficiently answered by simply traversing the M-Tree: at each non-leaf node storing a routing object O_r, the evaluation of both $dist(t,O_r)$ and $r(O_r)$ allows to decide whether the corresponding subtree $T(O_r)$ contains candidate neighbors and, hence, whether it has to be explored or not. In other words, querying the M-Tree for the neighbors of a query tuple involves traversing the M-Tree and ignoring those subtrees, that are reputed uninteresting for the search purpose.

A neighbor-driven approach to duplicate detection would strongly benefit from the exploitation of an index structure such as the M-Tree, since the latter would, in principle, answer similarity queries with minimal processing time and I/O cost. Additionally, the M-tree has three features, that are highly desirable in a de-duplication setting. First, it is a paged, balanced, and dynamic secondary-memory structure, capable to index data sets from generic metric spaces. Second, similarity range and nearest-neighbor queries can be performed and results can be ranked with respect to a given query tuple. Third, query execution is optimized to reduce both the number of pages read and the number of distance computations. However, the empirical analysis in [32] revealed that, when several tuples exhibit heterogeneous syntactic representations, the magnitude of the covering radii increases (especially at higher levels) and, hence, most of the internal nodes of the M-Tree tend to correspond to quite heterogeneous groups of tuples. Therefore, a high number of levels, nearly linear in the number of distinct entities in the data collection, is required to suitably index and organize the original collection of tuples. Thus, since in the typical de-duplication scenario the number of entities is likely to be of the same order as the number of available data tuples, the cost of similarity search actually tends to be nearly linear in the number of the original tuples. Clearly, this negatively affects the performance of the M-Tree and makes the overall neighbor-driven de-duplication process unable to scale for manipulating very large collections of data.

The problem of finding all tuples similar to a certain query tuple has been intensively studied within the database and information-retrieval communities with important achievements. Unfortunately, the incorporation of these results in neighbor-driven de-duplication would not make its computational cost independent on the size of the available data.

Some works from the database community, such as [86] and [5], focused on solving the problem exactly, by defining *set-similarity joins*, i.e. suitable operators for database management systems. Informally, a similarity join is an operation for recognizing different representation of a real-world entity. More precisely, given two relations, a similarity join finds all pairs tuples from the two relations, that are syntactically similar. Similarity is evaluated by means of a string-based similarity function: two tuples are considered similar if the value of the similarity function for these two tuples is greater than a certain threshold. An important drawback of the operators in [86] is that they scale quadratically with respect to the size of the data, which makes their exploitation impractical in the de-duplication of very large databases. The notion of set-similarity join was introduced in [5] as a primitive that takes two collections of sets as input and identifies all pairs of sets exhibiting a strong similarity. The latter is established through suitable predicates concerning the size and overlap of the sets. Set-similarity joins are performed through signature-based algorithms. These algorithms generate signatures for the input sets, with the desirable property that, if the similarity of two sets exceeds a certain threshold, then the two sets share a common signature. By exploiting this property, signature-based schemes find all pairs of sets with common features and, eventually, output all those pairs of sets whose pairwise similarity actually trespasses some preestablished threshold. The algorithms developed in [5] for performing set-similarity joins improve the basic performance of signature-based algorithms in two respects: the adoption of a different scheme for computing set signatures as well as the incorporation of a theoretical guarantee, according to which two highly dissimilar sets are not considered as duplicates with a high probability. The latter property of set-similarity joins considerably lowers the overall number of false-positive candidate pairs and, also, increases the efficiency of the resulting operators, that scale almost linearly in the size of the input set. However, linear scalability comes at expense of a non trivial parameter tuning, since no single parameter setting is appropriate for all computations. In practice, for a fixed parameter setting, the operators still scale quadratically and some properties of the input data must be analyzed so that to establish an optimal tuning, that ensures linear scalability.

Recently, an approach inspired from information retrieval methods [10] proposed to scale exact join-set methods to large volumes of real-valued vector data. This work refines the basic intuition in [86] of dynamically building an inverted list index of the input sets with some major indexing and optimization strategies, mainly concerning how the index is manipulated to evaluate the (cosine) similarity between the indexed records and the query one.

As a concluding remark, despite its effectiveness and the recent efforts for improving its scalability, similarity search for duplicates is not always a feasible approach to neighbor-driven de-duplication. It was shown in [3, 4] that either the space or the time required by the solutions devoted to expedite similarity search is exponential in the dimensionality of the data[3]. Moreover, it was also proven in [94] that similarity search based on space-partitioning indexing-schemes degenerates into a sequential scan of the data, even when dimensionality is moderately high. Therein, theoretical and empirical achievements lead to postulate in [94] that all approaches to nearest-neighbor search ultimately become linear in the size of the data, when dimensionality is sufficiently high. In the context of neighbor-driven de-duplication, the degeneration of the search for neighbors of the query tuple to a linear scan is undesirable, especially when processing very large volumes of data tuples.

4.3 De-duplication Based on Locality-Sensitive Hashing

De-duplication based on locality-sensitive hashing overcomes the limited scalability of neighbor-driven de-duplication. The premise is that in many de-duplication scenarios there exist several tuples, that are predominantly dissimilar from one another. Therefore, the number of groups of duplicates is likely to be of the same order as the overall number of tuples. In this context, finding few tuples mostly similar to the query tuple (i.e., to the generic tuple to be de-duplicated) is deemed to provide enough information for assigning the latter to the most appropriate group of duplicates through voting mechanisms.

The foregoing arguments motivate the exploitation of approximated similarity search for nearest neighbors of the query tuple, which can be performed much faster than exact similarity search by means of a suitable hash-based indexing method, that expedites the overall de-duplication process. To elaborate, an index structure is used to allow direct access to subsets of tuples with the same features.[4] The assignment of each individual tuple to the buckets of the index structure is managed by means of suitable hashing schemes. Precisely, the buckets associated to one tuple are the values of some hash function(s) on the features of the tuple. De-duplication benefits from a typical situation of hashing known as *collision*: the higher the similarity between two tuples, the likelier it is that these share the same features and, consequently, that both are assigned the same hash values, thus falling within the same buckets of the underlying index structure. This permits to narrow the search for neighbors of the query tuple to a focused linear search for nearest neighbors among those tuples falling within the buckets associated to (the individual features of) the query tuple. Therefore, in the de-duplication of the latter, nearest neighbors can be retrieved by issuing, against the index structure, similarity queries for neighboring

[3] The dimensionality of a structured tuple is simply the number of attributes in its schema. Instead, the dimensionality of a textual sequence can be viewed, in principle, as the number of distinguishing string tokens, chosen to represent the textual sequence as a point in a multidimensional space, according to the vector space model [7].

[4] Features are essentially distinguishing properties of a tuple, that are also commonly referred to as indexing keys (e.g., q-grams [46], i.e., contiguous substrings of length q, can be adopted to define suitable features for strings). Various key-generation schemes exist, that operate according to both the nature of available data and the specific applicative requirements.

tuples with the same features as the query tuple. Hash-based indexing guarantees that such queries can be efficiently answered. In particular, *locality-sensitive* hashing allows an indexing method for approximated similarity search, with a sub-linear dependence on the number of tuples. The idea behind locality-sensitive hashing is to bound the probability of collisions to the similarity between the tuples. In other words, a locality-sensitive hash function guarantees that the probability of collisions is much higher for similar tuples than it is for dissimilar ones. An important family of locality-sensitive hash functions, in which the similarity of two tuples is measured by their degree of overlap, can be naturally defined through the theory of *min-wise independent permutations* [19].

Various approaches to de-duplication based on locality-sensitive hashing have been proposed in the literature [20, 32, 43, 57].

Locality-sensitive hashing was originally developed in [57], as an efficient technique for accurately approximating the nearest-neighbor search problem.

The approach in [43] refines the basic proposal of [57] in several respects, including new theoretical guarantees on the worst-case time required for performing a nearest neighbor search as well as the generalization to the case of external memory. In particular, given a set of tuples and a family $\mathscr{H}$ of locality-sensitive hash functions, the generic tuple is associated with a corresponding bitstring signature, that identifies an index bucket. The bitstring is obtained from the values of k locality-sensitive hash functions (randomly extracted from $\mathscr{H}$) over the tuple. Since the overall number of buckets identified by the resulting bitstrings can be huge, a second level of standard hashing is exploited to compress the foresaid buckets by mapping their contents into one hash table $\mathscr{T}$, whose buckets are directly mapped to blocks on disk. The size of $\mathscr{T}$ is proportional to the ratio of the number of tuples to the maximal bucket-size of $\mathscr{T}$. In general, it is possible to loose proximity relationships if a point and its nearest neighbor are hashed to distinct buckets. Therefore, in order to lower the probability of such an event, the same tuple is stored in [43] into l hash tables $\mathscr{T}_1,\ldots,\mathscr{T}_l$, respectively indexed by as many independent bitstrings.

When it comes to de-duplicate a query tuple, its nearest neighbors are identified through the technique in [43] as follows. The query tuple is hashed to the buckets within the individual hash tables $\mathscr{T}_1,\ldots,\mathscr{T}_l$. All tuples previously hashed within these same buckets are gathered as candidate neighbors. A linear search is then carried out across these candidates, to find the neighbors actually closest to the query tuple. These are guaranteed to be at a distance from the latter within a small error factor of the corresponding optimal neighbors.

The drawback of the technique in [43] is the requirement for the identification of an optimal, data-specific tradeoff between two contrasting aspects of the index [9], namely accuracy and storage space. By increasing l, accuracy is guaranteed for the great majority of queries, through a correspondingly larger number of hash tables. However, this makes the storage requirement inversely proportional to the error factor. Also, it raises the number of potential neighbors and, hence, the overall response time. In such cases, one may act on k, since a high value of this parameter would sensibly lower the number of collisions and, hence, mitigate the increase in response time. Unfortunately, large values of *k* augment the miss rate. By the converse, small values of parameter *l* cannot guarantee accuracy for all queries.

An approach for identifying near duplicate Web pages is proposed in [20]. Here, each Web page is first tokenized and then represented as the set of its distinct, contiguous *n*-grams (referred to as shingles). The most frequent shingles are removed from the set to both improve performance and avoid potential causes of false resemblance. After preprocessing, near duplicates are identified via a clustering strategy that consists of the following four steps. A sketch is computed for each Web page, by applying a suitable min-wise independent permutation to its shingle representation. Sketches are then expanded to generate a list of all the individual shingles and the Web pages they appear in. Subsequently, this list is exploited to generate a new list of all the pairs of Web pages with common shingles, along with the number of shared shingles. Clustering is eventually achieved by examining the triplet elements of the latter list. If a certain pair of Web pages exceeds a pre-specified threshold for resemblance (estimated by the ratio of the number of shingles they have in common to the total number of shingles between them), the two Web pages are connected by a link in a union-find algorithm, that outputs final clusters in terms of connected components.

The algorithm in [20] requires a considerable amount of time and space on disk, especially due to the third phase, which makes it unscalable. Optimizations based on the notion of super-shingle addressed such an aspect, although these do not properly work with short Web pages. Yet, the de-duplication process strictly requires that the resemblance threshold is very high to effectively prune several candidate pairs of similar Web pages. Lower values of the threshold, corresponding to a typical setting for similarity search, cause several negative effects. False positive candidates are not appropriately filtered, which lowers precision. Very low values of the similarity threshold may also diminish false negatives, with a consequently moderate increase in recall. However, in such cases, the impact on the effectiveness of a small gain in recall would be vanished by the corresponding (much larger) loss in precision.

An incremental clustering technique for duplicate detection in very large databases of textual sequences is proposed in [32]. The techniques works by assigning each newly arrived tuple to an appropriate cluster of duplicates. More precisely, the de-duplication of a new tuple is accomplished by retrieving a set of neighboring tuples from a hash-based index structure. Neighbors are highly similar to the new tuple and, hence, their cluster membership provides useful information about the real-world entity corresponding to the new tuple. The latter is eventually assigned to the cluster of duplicates shared by the majority of neighbors.

The de-duplication process in [32] relies on a suitable hash-based index, that maps any tuple to a set of indexing keys and assigns syntactically similar tuples to the same buckets. In this manner, the neighbors of a query tuple can be efficiently identified by simply retrieving those tuples stored within the same buckets assigned to the query tuple itself, without either completely scanning the original database or using costly similarity metrics. Indexing keys are computed through a two-step key-generation procedure, in which locality-sensitive hash functions (based on a family of practically min-wise independent permutations [19, 43]) are hierarchically combined for a twofold purpose: reflecting the syntactic differences both among tuples and their components as well as enabling effective on-line matching. The first step of the key-generation scheme recognizes similar tuple components (i.e. string tokens) across tuples, despite some extent of syntactic heterogeneity. For this purpose, the individual tuples are purged into intermediate representations. These are obtained by encoding each component of the generic tuple via a min-wise hash function, that bounds the probability of collisions of two string tokens to the overlap between their respective sets of 1-grams (i.e., substrings of unit length) [46]. The choice of such a hash function guarantees that two similar but different tokens are, with high probability, assigned a same encoding. Therefore, any two tuples sharing syntactically similar tokens are purged into two intermediate representations, where such tokens converge towards a unique encoding. The second step of the key-generation scheme associates the intermediate representations of the original tuples with their indexing keys, through another min-wise hash function. The latter bounds the probability of collisions of two intermediate tuple representations to the overlap between their first-step token encodings. Again, this guarantees that two intermediate tuple representations sharing several first-step token encodings are associated with a same indexing key.

As a matter of fact, multiple min-wise hash functions are exploited both at the first and at the second step, for a twofold purpose, i.e., lowering the probability of false positives and false negatives (that essentially allows for a controlled level of approximation in the search for the nearest neighbors of the query tuple) as well as gaining a direct control over the number of keys used for indexing any tuple, which is necessary to guarantee the compactness of the overall storage space.

Interestingly, the approach in [32] has connections with several techniques from the literature.

Foremost, the two-step procedure for hashing tuples can be viewed as a smarter implementation of canopies (which are collected within the same buckets in the index). The main difference is that the properties of min-wise hashing functions allow to approximately detect such canopies incrementally.

Also, the key-generation scheme allows a constant (moderate) number of disk writes and reads to/from the index structure on secondary memory, which are two key aspects. Indeed, on one hand, the hash-based approach to de-duplication could cause continuous leaps in the disk-read operations, even when a small number of comparisons is needed for retrieving the neighbors of the query tuple. On the other hand, the number of disk pages written while updating the index structure is especially relevant in the incremental maintenance of the latter. Notably, the constant number of disk writes and reads is achieved in [32] along with a fixed (low) rate of false negatives. These are likely contrasting objectives in hash-based approaches to de-duplication, since few indexing keys must generally be produced to lower the amount of I/O operations, although this tends to increase the rate of false negatives.

Yet, from a methodological point of view, the approach in [32] exhibits analogies with the ones in [10, 86]. Indeed, a hash-based index is employed to maintain associations between a certain tuple feature and the subset of all available tuples that share that same feature. The actual difference with respect to the techniques in [10, 86] is that these approaches essentially pre-compute the nearest neighbors of each tuple, so that retrieving them becomes a simple lookup. Instead, the approach in [32] does not support neighbor pre-computation. Notwithstanding, it still enables neighbor search in a time that, on average, is independent on the number of database tuples.

Finally, the tradeoff between accuracy and storage space in [32] is much less challenging than in [43], since there exists a single hash-based index. Moreover, guaranteeing accuracy for all queries in [32] (i.e., lowering both the false-positive and false-negative rates) can be simply achieved by hashing a same tuple into as many buckets of the index as the number of first-step encodings of the tuple itself (for each such a representation, the concatenation of its second-step encodings yields the hash key associated to the tuple). Actually, a very limited number of distinct hash functions is used both at the first and at the second step to enforce de-duplication effectiveness, at the acceptable cost of a compact storage space.

5 Conclusions and Further Research

Duplicate detection is a necessary building block both for information integration and for the design of information systems, that allows more effective information handling and knowledge extraction, through a unified access and manipulation of consolidated and reconciled data. The current practices behind state-of-the-art approaches to duplicate detection have been discussed so far. Despite the considerable advances in the field, there are still various opportunities for further improvements [41, 47, 58, 98]. An overview of some major open problems and challenges is provided below.

5.1 Efficiency and Scalability

The detection of duplicates in very large volumes of data involves a huge number of pairwise tuple comparisons. In turn, each tuple comparison encompasses several matchings between the individual tuple components. A meaningful reduction of the number and the cost of both types of comparisons is key to devise solutions with which to improve the efficiency and scalability of de-duplication. Therein, various strategies have been proposed: blocking [58], sorted neighborhood [54], canopies [72], similarity joins [28] are some proposals for reducing comparisons between pairs of tuples without penalizing de-duplication effectiveness, whereas expedients such as [93] aim to speed up the individual comparison between tuple components. However, despite all such developments, efficiency and scalability often contrast with effectiveness. Indeed, the current state-of-the-art techniques can be divided into two broad classes [58], that highlight the foresaid contrast. Precisely, the techniques relying upon results from the field of databases privilege efficiency and scalability over effectiveness. On the contrary, other techniques exploiting contributions from the areas of machine learning and statistics reveal to be more effective. Unfortunately, their reduced efficiency and scalability makes the application of such techniques better suited for collections of data, whose size is orders of magnitude smaller than the collections processable through the techniques of the former type.

A worthwhile opportunity of further research is to devise new approaches combining the strengths of both types of de-duplication techniques.

5.2 Systematic Assessment of De-duplication Performance

Although several approaches to data de-duplication have been proposed in the literature, there is not yet a comprehensive and systematic knowledge of their relative performances in terms of de-duplication effectiveness as well as efficiency, across various data collections and domains. Recently, some efforts have been geared towards the design of evaluation frameworks [63, 64, 65], wherein to compare some de-duplication techniques. However, the process of gaining a systematic insight into the actual performances of de-duplication techniques is still in its infancy. The performed comparisons take into account only a limited number of approaches, that are compared on few collections of data from few real-world domains. Additionally, some collections of data were not effectively processed in [64]. This deserves further insights, that may be useful for the development of more sophisticated approaches.

Additionally, further research efforts for the identification of suitable performance metrics and state-of-the-art approaches across real-world domains should be fostered by the recent release of standard benchmarking data sets as the ones from [33]. Hopefully, the increasing availability of standard data sets will shed light on which metrics and techniques are best suited for the various applicative domains.

5.3 Ethical, Legal and Anonymity Aspects

Duplicate detection raises ethical, legal and privacy issues, whenever dealing with sensitive information about persons [62, 80]. Meeting the related national and international laws in force is thus a major requirement in the process of data de-duplication and subsequent management. A wide discussion concerning some major issues in data de-duplication and the mechanisms for the protection of individual privacy can be found in [90]. Anonymity preservation is one challenging issue, that is related to the release of large data collections for research and analytical purposes. The public release of any collections of data demands balancing analytic and research requirements with anonymity and confidentiality. Currently, a commonly adopted expedient to maintain the anonymity of persons in the underlying data is encrypting the identifying information. However, encryption may decrease the effectiveness of duplicate detection over time [78]. Another strategy for anonymity preservation would be the focused removal of the information necessary for tuple identification (and, hence, duplicate detection) from the available data sets. Unfortunately, this would still involve trust in some suitable preprocessing of the data [62].

One promising line of research for anonymity and confidentiality preservation is the development of ad-hoc models, with which to accurately assess identifying information leakage in the context of duplicate detection [41]. This would be useful to evaluate the leakage consequent to the release of identifying or sensible information and may be adopted as a criterion for carrying out the necessary precautions, e.g., global recoding and local suppression, additive noise and data perturbation as well as micro-aggregation (see [98] for a discussion of such precautions). Also, it would be very useful to incorporate models of analytic properties and re-identification risk in the process of de-duplication to answer two research questions, namely, understanding how the foresaid precautions compromise the analytic validity of the released data and estimating the re-identification rate if analytic validity is maintained.

5.4 Uncertainty

Uncertainty plays an important role in duplicate detection and is thus necessary to deal with it. Therein, a proposal is to attach confidences to the individual tuples. Confidences are essentially quantitative beliefs, whose actual meaning of confidences depends on the targeted applicative settings. In certain domains, confidences can be interpreted as the probability that the related tuples faithfully correspond to certain entities [41]. In other settings, confidences can be viewed as measures of the accuracy of the data.

While some efforts have been performed to efficiently compute the confidence of the de-duplicated tuples when confidences are explicitly attached to the tuples [73], the indirect estimation of some sort of de-duplication confidence, when uncertainty is not originally available, appears to be still unexplored.

References

1. Agichtein, E., Ganti, V.: Mining Reference Tables for Automatic Text Segmentation. In: Proc. of ACM SIGKDD Int. Conf. On Knowledge Discovery and Data Mining, Seattle, Washington, USA, pp. 20–29 (2004)
2. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating Fuzzy Duplicates in Data Warehouses. In: Proc. of Int. Conf. on Very Large Databases, Hong Kong, China, pp. 586–597 (2002)
3. Andoni, A., Indyk, P.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In: Proc. of IEEE Symposium on Foundations of Computer Science, Las Vegas, Nevada, USA, pp. 459–468 (2006)
4. Andoni, A., Indyk, P.: Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. Communications of the ACM 51(1), 117–122 (2008)
5. Arasu, A., Ganti, V., Kaushik, R.: Efficient Exact Set-Similarity Joins. In: Proc. of Int. Conf. on Very Large Databases, Seoul, Korea, pp. 918–929 (2006)
6. Axford, S.J., Newcombe, H.B., Kennedy, J.M., James, A.P.: Automatic Linkage of Vital Records. Science 130, 954–959 (1959)
7. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley, Reading (1999)

8. Bansal, N., Blum, A., Chawla, S.: Correlation Clustering. Machine Learning 56(1-3), 89–113 (2004)
9. Bawa, M., Tyson Condie, S., Ganesan, P.: LSH Forest: Self-Tuning Indexes for Similarity Search. In: Proc. of Int. Conf. on World Wide Web, Chiba, Japan, pp. 651–660 (2005)
10. Bayardo, R.J., Srikant, R., Ma, Y.: Scaling Up All Pairs Similarity Search. In: Proc. of Int. Conf. on World Wide Web, Banff, Alberta, Canada, pp. 131–140 (2007)
11. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB Journal 18(1), 255–276 (2009)
12. Berson, T.A.: Differential Cryptanalysis Mod 232 with Applications to MD5. In: Proc. of Ann. Conf. on Theory and Applications of Cryptographic Techniques, pp. 71–80 (1992)
13. Bhattacharya, I., Getoor, L.: Collective Entity Resolution in Relational Data. ACM Trans. Knowl. Discovery from Data 1(1), 1–35 (2007)
14. Bhattacharya, I., Getoor, L., Licamele, Louis: QueryTime Entity Resolution. In: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Philadelphia, Pennsylvania, USA, pp. 529–534 (2006)
15. Bilenko, M., Mooney, R.J.: Adaptive Duplicate Detection Using Learnable String Similarity Measures. In: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Washington, DC, USA, pp. 39–48 (2003)
16. Christen, P.: Towards Parameter-free Blocking for Scalable Record Linkage. Tech. Rep. TR-CS-07-03, Australian National University, Canberra, Australia (2007)
17. Christen, P.: Febrl - An Open Source Data Cleaning, Deduplication and Record Linkage System with a Graphical User Interface. In: Proc. of ACM Int. Conf. on Knowledge Discovery and Data Mining, pp. 1065–1068 (2008)
18. Borkar, V.R., Deshmukh, K., Sarawagi, S.: Automatic Segmentation of Text into Structured Records. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, Santa Barbara, California, USA, pp. 175–186 (2001)
19. Broder, A., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Minwise Independent Permutations. In: Proc. of ACM Symposium on Theory of Computing, Dallas, Texas, USA, pp. 327–336 (1998)
20. Broder, A., Glassman, S., Manasse, M., Zweig, G.: Syntactic Clustering on the Web. In: Proc. of Int. Conf. on World Wide Web, Santa Clara, California, USA, pp. 1157–1166 (1997)
21. Cesario, E., Folino, F., Locane, A., Manco, G., Ortale, R.: Boosting Text Segmentation Via Progressive Classification. Knowl. and Inf. Syst. 15(3), 285–320 (2008)
22. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and Efficient Fuzzy Match for Online Data Cleaning. In: Proc. of ACM SIGMOD Conf. on Management of Data, San Diego, California, USA, pp. 313–324 (2003)
23. Chaudhuri, S., Ganti, V., Motwani, R.: Robust Identification of Fuzzy Duplicates. In: Proc. of Int. Conf. on Data Engineering, Tokyo, Japan, pp. 865–876 (2005)
24. Chavez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Searching in Metric Spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
25. Ciaccia, P., Patella, M., Zezula, P.: M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: Proc. of Int. Conf. on Very Large Databases, Athens, Greece, pp. 426–435 (1997)
26. Cochinwala, M., Dalal, S., Elmagarmid, A.K., Verykios, V.S.: Record Matching: Past, Present and Future. Technical Report, number CSD-TR #01-013. Department of Computer Sciences, Purdue University (2001)

27. Cochinwala, M., Kurien, V., Lalk, G., Shasha, D.: Efficient Data Reconciliation. Information Sciences 137(1-4), 1–15 (2001)
28. Cohen, W.W.: Data Integration using Similarity Joins and a Word-based Information Representation Language. ACM Trans. on Inf. Syst. 18(3), 228–321 (2000)
29. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: Proc. of IJCAI Workshop on Information Integration on the Web, Acapulco, Mexico, pp. 73–78 (2003)
30. Cohen, W.W., Richman, J.: Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration. In: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, pp. 475–480 (2002)
31. Cohn, D.A., Atlas, L., Ladner, R.E.: Improving Generalization with Active Learning. Machine Learning 15(2), 201–221 (1994)
32. Costa, G., Manco, G., Ortale, R.: An Incremental Clustering Scheme for Data De-duplication. Data Min. and Knowl. Discovery 20(1), 152–187 (2010)
33. Database Group Leipzig. Benchmark datasets for entity resolution, `http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution`
34. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society, Series B 39(1), 1–28 (2001)
35. Dittrich, J.-P., Seeger, B.: Data Redundancy and Duplicate Detection in Spatial Join Processing. In: Proc. of IEEE Int. Conf. on Data Engineering, pp. 535–546 (2000)
36. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate Record Detection: A Survey. IEEE Transanctions on Knowledge and Data Engineering 19(1), 1–16 (2007)
37. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: Proc. of Int. Conf. on Knowledge Discovery and Data Mining, Portland, Oregon, USA, pp. 226–231 (1996)
38. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Widener, T.: The KDD Process for Extracting Useful Knowledge from Volumes of Data. Communications of the ACM 39(11), 27–34 (1996)
39. Fellegi, I.P., Sunter, A.B.: A Theory for Record Linkage. Am. Stat. Assoc. 64, 1183–1210 (1969)
40. Ganti, V., Ramakrishnan, R., Gehrke, J., Powell, A.: Clustering Large Datasets in Arbitrary Metric Spaces. In: Proc. of Int. Conf. on Data Engineering, Sydney, Austrialia, pp. 502–511 (1999)
41. Garcia-Molina, H.: Entity resolution: Overview and challenges. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 1–2. Springer, Heidelberg (2004)
42. Gilks, W.R., Richardson, S., Spiegelhalter, D.J.: Markov Chain Monte Carlo in Practice. Chapman and Hall, Boca Raton (1996)
43. Gionis, A., Indyk, P., Motwani, R.: Similarity Search in High Dimensions via Hashing. In: Proc. of Int. Conf. on Very Large Databases, Edinburgh, Scotland, pp. 518–529 (1999)
44. Goiser, K., Christen, P.: Towards Automated Record Linkage. In: Proc. of Australasian Data Mining Conf., pp. 23–31 (2006)
45. Grabmeier, J., Rudolph, A.: Techniques of Cluster Algorithms in Data Mining. Data Min. and Knowl. Discovery 6(4), 303–360 (2002)
46. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate String Joins in a Database (Almost) for Free. In: Proc of Int. Conf. on Very Large Databases, Rome, Italy, pp. 491–500 (2001)

47. Gu, L., Baxter, R.A., Vickers, D., Rainsford, C.: Record Linkage: Current Practice and Future Directions. Technical Report, number 03/83. CSIRO Mathematical and Information Sciences (2001)
48. Guha, S., Rastogi, R., Shim, K.: CURE: An Efficient Clustering Algorithm for Large Databases. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, Seattle, Washington, USA, pp. 73–84 (1998)
49. Guha, S., Rastogi, R., Shim, K.: ROCK: A Robust Clustering Algorithm for Categorical Attributes. Inf. Syst. 25(5), 345–366 (2001)
50. Gunsfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Davis (1997)
51. Hassanzadeh, O., Chiang, F., Lee, H.C., Miller, R.J.: Framework for Evaluating Clustering Algorithms in Duplicate Detection. Proceedings of VLDB 2(1), 1282–1293 (2009)
52. Hassanzadeh, O., Miller, R.J.: Creating Probabilistic Databases from Duplicated Data. The VLDB Journal 18(5), 1141–1166 (2009)
53. Hernández, M.A., Stolfo, S.J.: The Merge/Purge Problem for Large Databases. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, San Jose, California, USA, pp. 127–138 (1995)
54. Hernández, M.A., Stolfo, J.: Real-world Data is Dirty: Data Cleansing and the Merge/Purge Problem. Data Min. and Knowl. Discovery 2(1), 9–37 (1998)
55. Herschel, M., Naumann, N.: Scaling up Duplicate Detection in Graph Data. In: Proc. of ACM Int. Conf. on Information and Knowledge Management, pp. 1325–1326 (2008)
56. Hjatason, G.R., Samet, H.: Index-Driven Similarity Search in Metric Spaces. ACM Trans. on Database Syst. 28(4), 517–518 (2003)
57. Indyk, P., Motwani, R.: Approximate Nearest Neighbor - Towards Removing the Curse of Dimensionality. In: Proc. of Symposium on Theory of Computing, Dallas, Texas, USA, pp. 604–613 (1998)
58. Ipeirotis, P.G., Verykios, V.S., Elmagarmid, A.K.: Duplicate Record Detection: A Survey. IEEE Trans. Knowl. Data Eng. 19(1), 1–16 (2007)
59. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs (1998)
60. Jain, A.K., Murty, M.N., Flynn, P.J.: Data Clustering: A Review. ACM Comput. Surv. 31(3), 264–323 (1999)
61. Jaro, M.A.: Advances in Record Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. Journal of the American Statistical Society 84, 420–424 (1989)
62. Kingsbury, N.R., et al.: Record Linkage and Privacy: Issues in Creating New Federal Research and Statistical Information. U.S. General Accounting Office (2001)
63. Kopcke, H., Rahm, E.: Frameworks for Entity Matching: A Comparison Data and Know. Engineering 69(2), 197–210 (2010)
64. Kopcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. Proc. of the VLDB Endowment 3(1), 484–493 (2010)
65. Kopcke, H., Thor, A., Rahm, E.: Evaluation of Learning-Based Approaches for Matching Web Data Entities. IEEE Internet Computing 14(4), 23–31 (2010)
66. McCallum, A.: MALLET: A Machine Learning for Language Toolkit, `http://mallet.cs.umass.edu`
67. Koudas, N., Sevcik, K.C.: Size Separation Spatial Join. In: Proc. of ACM Int. Conf. on Management of Data, pp. 324–335 (1997)
68. Lawrence, S., Bollacker, K., Giles, C.L.: Autonomous Citation Matching. In: Proc. of ACM Int. Conf. on Autonomous Agents, pp. 392–393 (1999)

69. Low, W.L., Lee, M.L., Ling, T.W.: A Knowledge-Based Approach for Duplicate Elimination in Data Cleaning. Information Systems 26(8), 585–606 (2001)
70. Lwin, T., Nyunt, T.T.S.: An Efficient Duplicate Detection System for XML Documents. In: Proc. of IEEE Int. Conf. on Computer Engineering and Applications, pp. 178–182 (2010)
71. McCallum, A., Freitag, D., Pereira, F.: Maximum Entropy Markov Models for Information Extraction and Segmentation. In: Proc. of Int. Conf. on Machine Learning, Standord, California, USA, pp. 591–598 (2000)
72. McCallum, A., Nigam, K., Ungar, L.: Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Boston, Massachusetts, USA, pp. 169–178 (2000)
73. Menestrina, D., Benjelloun, O., Garcia-Molina, H.: Generic Entity Resolution with Data Confidences. In: Int. VLDB Workshop on Clean Databases, Seoul, Korea (2006)
74. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
75. Monge, A.E., Elkan, C.P.: An Efficient Domain-Independent Algorithm For Detecting Approximately Duplicate Database Records. In: Proc. of SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Tucson, Arizona, USA, pp. 23–29 (1997)
76. Monge, A.E., Elkan, C.P.: The Field Matching Problem: Algorithms and Applications. In: Proc. of Int. Conf. on Knowledge Discovery and Data Mining, Portland, Oregon, USA, pp. 267–270 (1996)
77. Mukherjee, S., Ramakrishnan, I.V.: Taming the Unstructured: Creating Structured Content from Partially Labeled Schematic Text Sequences. In: Proc. of CoopIS/DOA/ODBASE Int. Conf., Agia Napa, Cyprus, pp. 909–926 (2004)
78. Muse, A.G., Mikl, J., Smith, P.F.: Evaluating the quality of anonymous record linkage using deterministic procedures with the New York State AIDS registry and a hospital discharge file. Statistics in Medicine 14, 499–509 (1995)
79. Neiling, M., Jurk, S.: The Object Identification Framework. In: Proc. KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation, Washington, DC, USA, pp. 37–39 (2003)
80. Neutel, C.I.: Privacy Issues in Research Using Record Linkage. Pharmcoepidemiology and Drug Safety 6, 367–369 (1997)
81. Newcombe, H.B.: Record Linking: The Design of Efficient Systems for Linking Records into Individual and Family Histories. American Journal of Human Genetics 19, 335–359 (1967)
82. Newcombe, H.B., Kennedy, J.M., Axford, S.J., James, A.P.: Automatic Linkage of Vital Records. Science 130, 954–959 (1959)
83. Patel, J., DeWitt, D.J.: Partition Based Spatial-Merge Join. In: Proc. of ACM Int. Conf. on Management of Data, pp. 259–270 (1996)
84. Pasula, H., Marthi, B., Milch, B., Russell, S.J., Shpitser, I.: Identity Uncertainty and Citation Matching. In: Proc. of Ann. Conf. on Neural Information Processing Systems, pp. 1401–1408 (2002)
85. Sarawagi, S., Bhamidipaty, A.: Interactive Deduplication using Active Learning. In: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, pp. 269–278 (2002)
86. Sarawagi, S., Kirpal, A.: Efficient set joins on similarity predicates. In: Proc. of SIGMOD Int. Conf. on Management of Data, Paris, France, pp. 743–754 (2004)
87. Shen, H., Zhang, Y.: Improved Approximate Detection of Duplicates for Data Streams over Sliding Windows. Journal of Computer Science and Technology 23(6), 973–987 (2008)

88. Singla, P., Domingos, P.: Multi-Relational Record Linkage. In: Proc. of ACM Int. Ws. on Multi-Relational Data Mining, pp. 31–38 (2004)
89. Smith, S., Waterman, M.S.: Identification of Common Molecular Subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
90. Statistical Linkage Key Working Group. Statistical Data Linkage in Community Services Data Collections (2002)
91. Tejada, S., Knoblock, C.A., Minton, S.: Learning Domain-Independent String Transformation Weights for High Accuracy Object Identification. In: Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, pp. 350–359 (2002)
92. Tepping, J.B.: A Model for Optimum Linkage of Records. Journal of the American Statistical Association 63, 1321–1332 (1968)
93. Verykios, V.S., Elmagarmid, A.K., Houstis, E.N.: Automating the approximate record-matching process. Inf. Sci. 126(1-4), 83–98 (2000)
94. Weber, R., Schek, H.J., Blott, S.: A Quantitative Analsysis and Performance Study for Similarity Search in High-Dimensional Spaces. In: Proc. of Int. Conf. on Very Large Databases, New York City, USA, pp. 194–205 (1998)
95. Weis, M., Naumann, N.: Detecting Duplicates in Complex XML Data. In: Proc. of IEEE Int. Conf. on Data Engineering, p. 109 (2006)
96. Weis, M., Naumann, N.: Space and Time Scalability of Duplicate Detection in Graph Data. Tech. Rep. 25, Hasso-Plattner Institut, Potsdam, Germany (2007)
97. Winkler, W.E.: String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. In: Proc. Section on Survey Research Methods, American Statistical Association, pp. 354–359 (1990)
98. Winkler, W.E.: Overview of Record Linkage and Current Research Directions. Technical Report. Statistical Research Division, U.S. Census Bureau (1999)
99. Winkler, W.E.: Methods for Record Linkage and Bayesian Networks. Tech. Rep. RRS2002/05, U.S. Bureau of the Census, Washington, D.C., USA (2002)
100. Zhang, Y., Lin, X., Yuan, Y., Kitsuregawa, M., Zhou, X., Yu, J.X.: Duplicate-insensitive Order Statistics Computation over Data Streams. IEEE Transanctions on Knowledge and Data Engineering 22(4), 493–507 (2010)