UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica
XIX ciclo

Tesi di Dottorato

Entity Resolution: Effective Schema and Data Reconciliation

Francesco Folino

UNIVERSITÀ DELLA CALABRIA
Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XIX ciclo

Tesi di Dottorato

Entity Resolution: Effective Schema and Data Reconciliation

Francesco Folino

[illegible]

Coordinatore
Prof. Domenico Talia

[illegible]

Supervisore
Prof. Domenico Saccà

[illegible]

Dott. Giuseppe Manco

[illegible]

DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA
Settore Scientifico Disciplinare: ING-INF/05

To my beloved grandmother and to my father

Abstract

Integrating heterogeneous data in a Data Warehouse is a crucial step for extracting knowledge able to support critical decisions. Here, the resolution of systematic differences and conflicts pertaining both the structure and the values of data is essential. In particular, recognizing that two syntactically different tuples, with or without identical schema, refer to the same real-world entity is one of the most important integration issues, hardly affecting the quality of consolidated data. This problem is known in literature with the name of *Entity Resolution*.

The objective of this thesis is to define a complete path towards the resolution of entities by facing two main tasks: *(i) Schema Reconciliation*, consisting in the identification of a common field structure for the information extracted from a generic data source, and *(ii) Data Reconciliation*, that is the act of discovering synonymies (i.e., duplicates) in the data. In particular, we intend to deal with Entity Resolution in a more complicate scenario where very large volumes of data are involved. In this case, efficiency and scalability become not negligible considerations, thus imposing severe constraints on the design of data structures and algorithms for Entity Resolution.

Under the described setting, we propose three approaches dealing with both schema and data reconciliation. We first provide a rule-based, supervised classifier able to reconcile, in a fixed attribute schema, information extracted from unstructured data source as free text. Then, we illustrate the way for efficiently solving the synonymies in the data, by exploiting a scalable and incremental clustering approach, which core is the usage of a suitable *hash-based indexing technique* tailored to the *Jaccard similarity.* Finally, we overcome some drawbacks of the previous hash-based schema, by exploiting a more refined key-generation technique obtained by resorting to a family of *locality-sensitive hashing functions (LSH).*

For all the proposed approaches, an extensive experimental evaluation both on synthetic and real datasets, shows the effectiveness and efficiency of our proposals when compared against some state-of-the-art techniques in literature.

Acknowledgments

This thesis results from an intense research activity within the ICAR-CNR institute of Cosenza, whose members I owe to pay a tribute to. In particular, I wouldn't have been able to develop this thesis without the supervision of Dr. Giuseppe Manco. Also, I would like to thank my advisor, Prof. Domenico Saccà, for his kindness, encouragements and helpful suggestions during my Ph.D. years.
Special thanks go to my mother, for her loving support and patience, and for having always respected all decisions I made.
Moreover, I cannot forget to thank my friends and colleagues at ICAR-CNR institute. In particular, my brotherly friend Eugenio Cesario who shared with me enjoinments, anguishes and disappointments over these years.
Last but not least, I would like to thank other members of my family: my aunt Lina, my uncle Alberto, my cousins Anna and Francesca, and don Adamo, for their untiring support during all my life.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivations

The transformation of disparate data sources into knowledge to support critical decisions is essential in today's fast-growing market. Information sharing among organizations can help achieve important public benefits such as increased productivity, improved policy-making, and integrated public services. A large number of data sources are available in organizations in a variety of formats, such as flat-text files or databases. Moreover, all these data can be derived from different sources either in one organization or across multiple organizations.

The analysis of available data often influences important business decisions, and thus, its accuracy becomes a crucial aspect [49]. To this purpose, it is usually required to consolidate information from heterogeneous sources in a *Data Warehouse*: a historical, summarized collection of data aimed at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions.

By definition, a Data Warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making". Typically, a data warehouse is maintained separately from the organization's operational databases. There are many reasons for doing this: the data warehouse supports on-line analytical processing (OLAP), the functional and performance requirements of which are quite different from those of the on-line transaction processing (OLTP) applications traditionally supported by the operational databases.

However, in the act of integrating data from different data sources to implement a data warehouse, organizations become aware of potential systematic differences or conflicts. Data quality can be compromised by many factors: data entry errors, missing integrity constraints and multiple conventions for recording information. To make things worse, in independently managed databases, not only the values but even the structure, semantics and underlying assumptions about the data may differ as well.

Such problems fall under the umbrella-term *data heterogeneity* [23], *data cleaning* [124], or *data scrubbing* [152]. More in detail, the heterogeneity in the data can be distinguished into *structural* and *lexical*:

- *Structural heterogeneity* occurs when the fields of the tuples are structured differently in different databases. For instance, in one database, the customer address might be recorded in one field named *addr*, while in another database the same information might be stored in multiple fields such as *street*, *city*, *state*, and *zipcode*. A more general version of this problem deals with the integration of unstructured information (such as free text) in structured databases.
- *Lexical heterogeneity* occurs when the tuples have identically structured fields across databases, but data use different representations to refer to the same real-world object. For instance, *StreetAddress*= "44 E. 5th Street" vs. *StreetAddress* = "44 East Fifth Street".

Unfortunately, when information from disparate information sources must be combined, it is required to deal with both such heterogeneities. The task able to accomplish such a challenge is known with the name of *Entity Resolution*. The notion of *Entity Resolution* [31, 67, 153] denotes a complex, multi-steps process for these information integration problems that embraces two main tasks:

- *Schema Reconciliation* consists in the identification of a common field structure for the information in a generic data source.
- *Data Reconciliation* is the act of discovering synonymies (duplicates) in the data, i.e., apparently different records that, as a matter of fact, refer to a same real-world entity.

Typically, the Schema Reconciliation step is preparatory for the next Data Reconciliation step and thus may strictly affect its effectiveness.

Interest in Entity Resolution has grown rapidly in recent years, side by side with its applicability to a wide range of cases spanning from classical databases area to WWW. As witness of such an interest, the Entity Resolution problem has been studied across several research areas in which it has also taken multiple names: *record linkage* or *record matching* problem [109, 108, 143, 50, 107] in the statistics community. *Merge-purge* [74], *data de-duplication* [125], and *instance identification* [149] in the database community. In the AI community, it is described as *database hardening* [35] and *name matching* [12]. Moreover, the names *identity uncertainty*, and *duplicate detection* are also commonly used to refer the same task.

Many of the approaches in literature deal with the Entity Resolution problem just from an accuracy viewpoint. However, efficiency and scalability issues tend to assume a predominant role in all the contexts where very large volumes of data are involved. For instance, consider a banking scenario where information collected on daily basis typically consists of 500,000 instances, representing credit transactions performed by customers throughout various agencies. These large collections of data inevitably impose severe constraints to be considered on the the design of data structures and algorithms for Entity Resolution.

Under the described scenario, the goal of this thesis is to provide a complete path towards the resolution of entities by facing both the schema and data reconciliation steps. In this sense, its contribution is twofold:

1. providing a technique able to reconcile in a fixed attribute schema information extracted from an unstructured data source as text.
2. solving the synonymies in the tuples by exploiting a scalable and incremental approach able to deal with large volumes of data.

1.2 Main Contributions of the Thesis

The subject of this thesis is a systematic study of the overall Entity Resolution process when it deals with large volumes of data.

The scenario in which we move is particular in this sense:

- the source of data to de-duplicate is unstructured, i.e., the tuples are stored as free text.
- the extracted tuples are identified as (small) sequences of strings, where the set of possible string is high. If the strings represent dimensions along which the information contained in a tuple is mapped, tuples can be considered as vectors in a high-dimensional space.
- the large volume of data involved imposes scalable and incremental approaches disqualifying each non-linear (in the number of tuples) algorithm.

Actually, that above described, it does not represents the only possible scenario. For instance, the *reference disambiguation* problem arises when entities in a database contain references (*links*) to other entities. This problems is strictly related to the problem of record de-duplication. The differences between the two can be intuitively viewed using the relational terminology as follows: while the de-duplication problem consists of determining when two records are the same, reference disambiguation corresponds to ensuring that references in a database point to the correct entities. Further, when we draw some extensions of this thesis (see Chapter 7), we will briefly describe a link-based approach that systematically exploits not only features but also relationships among entities for the purpose of improving data reconciliation.

Ultimately, the main contributions of this thesis can be summarized as in the following:

1. We cope with the Schema Reconciliation problem viewed as the typical information extraction task of segmenting (restructuring) in a fixed attribute schema tuples stored as free text. We first analyze the most important proposals in literature in this field, with a particular attention at

some state-of-the-art stochastic segmenting approaches: *Hidden Markov Models* [1, 15, 127], *Maximum Entropy Markov Models* [96] and *Conditional Random Fields* [88] representing our sparring partners in the evaluation phase. Our contribution consists in providing a new methodological approach in which a strict cooperation between ontology-based generalization and rule-based classification is envisaged, which allows to reliably associate terms in a free text with a corresponding semantic category. A key feature is the introduction of *progressive classification*, which iteratively enriches the available ontology, thus allowing to incrementally achieve accurate schema reconciliation. This ultimately differentiates our approach from previous works in the current literature, which adopt schemes with fixed background knowledge, and hence hardly adapt to capture the multi-faceted peculiarities of the data under investigation.

2. We introduce a general framework for formally characterizing the problem of discovering and merging duplicate objects, essentially in terms of a specific *clustering* problem. According to this framework, we then describe an efficient technique which is able to discover clusters containing duplicate tuples in an incremental way. The core of this approach is the usage of a suitable indexing technique tailored to a set-based distance function (*Jaccard distance*), which, for any newly arrived tuple, allows to efficiently retrieve the set of its most similar tuples in the database (and hence likely referring to the same real-world entity). This indexing schema is based on a *hashing* technique assigning highly similar objects to the same buckets on the basis of the sharing of a certain number of common relevant features. Our approach is compared with a different, state-of-the-art, indexing technique (*M-tree* [30]), which allows for searching in general metric spaces, thus demonstrating a considerable efficiency improvement at the cost of a limited accuracy loss.
3. We extend and improve the above hashing proposal in both effectiveness and efficiency. First, we gain a direct control over the number of features used for indexing any tuple, which is the major parameter that critically impacts on the overall cost of the approach. Moreover, we tune the approach to be less sensitive to little differences between tokens of duplicate tuples. To this purpose, still in the setting of previous indexing schema, we exploit a more refined key-generation technique which, for each tuple under consideration, guarantees a direct control on the degree of granularity needed to properly discover its actual duplicates. In particular, we resort to a family of *locality-sensitive hashing* functions (LSH) [77, 20, 62], which are guaranteed to assign any two objects to the same buckets with a probability which is directly related to their degree of mutual similarity.

1.3 Thesis Organization

We proceed as follows.

In Chapter 2 we discuss the Schema Reconciliation problem by placing it in the wide area of Information Extraction. We give a general description of the typical stages constituting an IE process, and we itemize some typical IE tasks. Then, we provide a detailed overview of the main current approaches in the IE scenario. A particular attention is devoted to techniques coping with a particular IE task that aims to segment continuous text in a reference attribute schema.

In Chapter 3 we present *RecBoost*, our supervised, rule-based approach to Schema Reconciliation problem. Here, we discuss the reasons for the introduction of a new text segmentation tool, and the peculiarities making them particularly tailored to the Entity Resolution setting. Moreover, we provide an intensive experimental evaluation aiming at demonstrating *(i)* the effectiveness of the basic rule-based classifier and *(ii)* the improvement in accuracy obtained by means of the adoption of *progressive classification*. We also compare our results with other state-of-the-art text segmentation systems.

In Chapter 4 we focus on Data Reconciliation problem. We provide an extensive overview of the approaches in literature organized as follows. We first briefly discuss the typical steps in the data cleaning/merging process, before the *duplicate record detection* phase starts. Next, we describe techniques used to match individual fields in a record, and techniques for matching records containing multiple fields. Moreover, we present methods for improving the efficiency of de-duplication phase, and finally, we review some commercial standard tools used in industry for data reconciliation purposes.

In Chapter 5 we introduce our framework that copes with the de-duplication essentially in terms of a specific incremental clustering problem. To this purpose, we present a suitable hash-based indexing technique able to efficiently deal with the duplicate record detection when large volumes of data are involved. Moreover, we compare our approach with different indexing structures proposed in literature on synthesized and real data, by showing a considerable improvement in efficiency essentially avoiding a wasteful reduction of accuracy.

In Chapter 6 we extend and improve, in both effectiveness and efficiency, the hash-based approach proposed in the previous chapter. More in detail, we present a more refined key-generation technique based on a family of *locality-sensitive hashing* functions, which, in a probabilistic manner, guarantee to assign duplicates to a same bucket by considering their similarity degree.

In Chapter 7 we summarize our contribution and highlight still open problems that are worth further investigations.

2

Approaches to Schema Reconciliation

2.1 Introduction

As introduced in Chapter 1, *Entity Resolution* process consists of two tasks: *Schema Reconciliation*, and *Data Reconciliation*. In this chapter, we are interested to review some well-known solutions for the *Schema Reconciliation* problem when it deals with unstructured or loosely structured text.

Nowadays, several useful data sources exist as continuous text, primarily because humans find easier to create them that way. Examples are postal addresses, bibliography records, classified ads, and phone lists. Such sources could be more effectively queried and analyzed when imported into a structured relational table. Building and maintaining large data warehouses by integrating data from sources requires automatic conversion of text into structured records of the target schema before loading them into relations. This task is typically called *Information Extraction* (IE).

Starting from the 1980's, information extraction research has been largely encouraged and driven by the series of Message Understanding Conference (MUCs). The MUC extraction tasks ranged from parsing naval dispatches (MUC-1), to terrorist attacks in Latin America (MUC-3), to joint ventures and company acquisitions (MUC-5) as well as corporate management (MUC-7).

In general, information extraction refers to automatic methods for creating a structured representation of selected information drawn from natural language text. More specifically, information extraction systems can identify particular types of entities (such as drug names) and relationships between entities (such adverse interactions between medical drugs) in natural language text for storaging and retrieving purposes in a structured database [56]. In particular, if the input of the IE process is a string obtained by concatenating structured elements with limited reordering and some missing fields (e.g. addresses, bib records), the IE task can also be referred as *text segmentation*.

Informally, the problem of segmenting input strings into a structured record with a given n-attribute schema is to partition the string into n contiguous sub-strings and to assign each sub-string to a unique attribute of the schema. For instance, segmenting the input string “Mining reference tables for automatic text segmentation E. Agichtein, V. Ganti KDD” into a bibliographic record with latent schema $[Authors, Title, Conference, Year]$ requires the assignment of the sub-string “E. Agichtein, V. Ganti” to the Authors attribute, the sub-string “Mining reference tables for automatic text segmentation” to the Title attribute, “KDD” to the Conference attribute, and the NULL value to the Year attribute.

Current techniques for the IE task can be broadly classified into *rule-based* and *stochastic* approaches.

This chapter is organized as follows: in the Section 2.2 the Information Extraction scenario is presented. In Section 2.3 rule-based approaches for IE are explained. More in details, in Section 2.3.1 hand-coded approaches whereas in the Section 2.3.2 rule-based learning techniques are surveyed. Finally, Section 2.4 shows state-of-the-art stochastic approaches, and in Section 2.5 some conclusions are drawn.

2.2 The Information Extraction problem

Information extraction has long been a rich subject of research. An information extraction system typically examines every document in a collection and attempts to extract “interesting” information (*concepts*) for a given tabular format, by filling each attribute of the relation from the text of the document. Following the MUC nomenclature, we define an *event* as a complex relation (with multiple arguments) between entities, and a *scenario* as the task of extracting events from the data source. Moreover, we shall refer to the final tabular output as a *template* (i.e., a template entity, a template relation, and a event or scenario template).

As an example of a typical IE task, consider extracting the *CompanyHeadquarters(Organization:ORGANIZATION, Location:LOCATION)* relation, which contains a tuple (o, l) if organization o has headquarters in location l. More precisely, the relation *CompanyHeadquarters* represents the specific semantic relationship between companies and their locations, while an *instance* of *CompanyHeadquarters* contains a set of tuples (i.e., pairs) of individual company and location entities related as specified above [117].

Figure 2.1 shows the basic stages in the extraction of a tuple from a document fragment. As one of the first stages of extraction, the input documents are typically passed through a *named-entity tagger*, which is able to recognize entities such as organizations, locations, and persons. Named-entity tagging is a well studied problem, with tools publicly available for the most common entity types [42]. These entities are potential values of attributes in the target relation. To find related entities, the tagged documents are generally processed by applying extraction patterns or IE rules in the *pattern matching step*. These patterns may be manually constructed or automatically learned

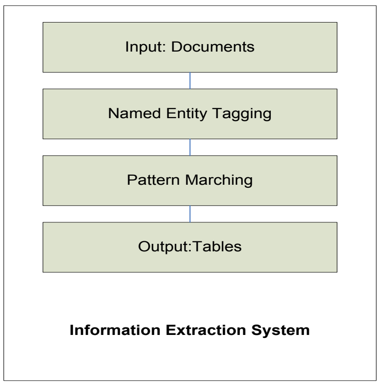


Fig. 2.1. Typical IE stages

(see Section 2.3). Each pattern is applied to each text fragment, instantiating appropriate slots in the pattern with entities from the document. These entities are combined into candidate tuples, and after filtering and post-processing are returned as extracted tuples.

The most general and complex extraction task is the event or scenario extraction. For instance, for a terrorist attack event, we expect to extract the time, location, perpetrator, target, damage, victims, etc. An event often spans several sentences, so we may need inference to figure out all the slots of an event. Many of rule-based systems are devoted to face this task.

Although rule-based techniques are the most common ones for IE, several approaches explore the use of well-known stochastic machine learning methods. We will discuss some of these approaches in Section 2.4.

2.3 Rule-based Approaches

In the typical IE setting, the rules useful for the extraction task are classified as *single-slot rules* and *multi-slot rules*, given that a concept can be represented as a template. A single-slot rule is able to extract document fragments related to one slot within a template, while a multi-slot rule extracts tuples of document fragments related to the set of slots within a template. As introduced above, these rules can be either manually-engineered or (the greater part) learned from large amount of text (annotated and otherwise) when available.

2.3.1 Hand-crafted Information Extraction Systems

At the beginning, IE systems were customized manually to a given task. Due to the very high manual effort for tuning these systems, the attention was focused on automating the IE approaches. Anyhow, the best performance are still obtained by hand-crafted systems. In the following, two of the most famous systems, in which the pattern recognition phase is completely hand-coded, are proposed.

FASTUS

FASTUS [6], developed in the early 1990s, is a (slightly permuted) acronym for *Finite State Automaton Text Understanding System*, although it does not deal with text understanding but IE. It is a cascaded, nondeterministic finite state automaton (FSA). More specifically, FSA's are used both for the phrasal decomposition and for the recognition of domain-specific phrases.

FASTUS performs the IE task in four steps:

1. *Triggering* takes place based on the search of a set of predefined keywords (trigger words) specific to each extraction pattern.
2. *Phrasal decomposition* takes a relevant sentence and breaks it into noun groups, name groups, verb groups, and several classes of critical words (e.g., prepositions, pronouns, conjunctions, etc.).
3. *Pattern recognition* is performed based on a predefined, completely hand-coded set of rules. For instance, "`killing of <victim>`" and "`Bomb was placed by <perpetrator> on <target>`" are example of valid FASTUS patterns.
4. the *incident merging* phase combines into a single incident the information extracted into several complementary incidents.

The basic system is relatively small, although the dictionary used is very large. The manually developed rules were very effective and performed very well.

PROTEUS

PROTEUS [159] is a core extraction engine consisting of seven modules: (1) lexical analysis, (2) name recognition, (3) partial syntactical analysis, (4) scenario pattern analysis, (5) reference resolution, (6) discourse analysis, and (7) output generation.

The *lexical analysis* module splits the document into sentences and tokens. Each token is assigned a reading using dictionaries. Optionally, a parts-of-speech tagger can be invoked to eliminate unlikely readings from tokens. The *name recognition*, *partial syntax*, and *scenario patterns* modules use deterministic, bottom-up, partial parsing, or pattern matching. Patterns are regular expressions. Patterns for name recognition identify proper names. The *partial syntax* module finds noun phrases and verb phrases. The *scenario patterns* module finds higher-level syntactic constructions. The *reference resolution* module links anaphoric pronouns or their antecedents and merges other co-referring expressions. The *discourse analysis* module builds more complex event structures using higher-level inference rules. Thereby, several clauses contain information about a single complex fact. The *template generation* module performs a transformation of the gathered information into the final template structure.

The pattern acquisition consists of several steps. First, the user enters a sentence containing an event and selects an available template. Then, the system applies current patterns to the example to obtain an initial analysis. Thereby, it identifies noun/verb groups and their semantic types and applies a minimal generalization. The system presents the result to the user, who can modify each pattern element (e.g., choose the appropriate level of generalization, make the element optional, remove it). The user then has to specify how pattern elements are used to fill slots in the template. Now the system builds a new pattern to match the example and compiles the associated action, which will fire when the pattern matches and will fill the slots of the template. The new pattern is added to the pattern base.

2.3.2 Learning-based Information Extraction Systems

A hand-crafted pattern is writing by domain knowledge experts in order to extract useful information from a particular domain. For instance, in the terrorism domain, a possible pattern would be formed as "`<target> was bombed`". This pattern could be applied onto the sentence "A public building was bombed in Oklahoma City last year", and "A public building" can be extracted as *target*. Thus, if we were informed beforehand about the event, event structure and its domain, an IE system, well customized for the particular target event structure, could be developed.

However, because of the wide availability of textual documents and the diversity of needs for information extraction, it seems impractical to develop an IE system for each domain. Thus, the focus of the studies of information extraction was concentred toward its portability across domains.

Hence, the motivation of exploiting learning-based models is to reduce human effort in building or shifting an IE system. Instead of creating patterns by hand, these models derive rules via generalization of examples. The learning process starts with patterns from specific examples (tagged text segments) and the syntactic structure of the surrounding text. Then, it tries to generalize the patterns inductively by relaxing constraints and merging patterns. This process continues until no more generalization can be done without introducing too many errors. The result is a set of generalized patterns.

Categorizing the learn-based approaches is a task not easy to fulfill. Such approaches are in literature classified from different points of view: the degree of supervision, the kind of rules learned, the learning paradigm (e.g., propositional or relational), and so on. In this section, some of state-of-the-art IE rule learning systems are classified according to these criteria. For the sake of simplicity, the degree of supervision they need has been taken as the starting point for the comparison.

Supervised Approaches

Under the inductive learning paradigm, supervised approaches are the most common ones for learning IE rules. In general, a learning method is supervised if some intervention is required from the user for the learning process. Some approaches require the user to provide training examples. This supervision can be carried out as a preprocess (i.e., appropriately tagging the examples occurring in the training set) or as an online process (i.e., dynamically tagging the examples as needed during the learning process). Some of these learning approaches focus on propositional learning (e.g., AutoSlog, PALKA, CHRYSTAL) while others focus on relational learning (e.g., SRV, RAPIER, WHISK).

Propositional learning is based on representing the examples of a concept in terms of either zero order logic or attribute-value logic. Relational learning is based on representing the examples of a concept in terms of first order logic.

- **AutoSlog**
  AutoSlog [119] generates single-slot rules, named *concept nodes*, by applying a heuristic-driven specialization strategy. An AutoSlog concept has three key componenents:
  - the *conceptual anchor* that activates it.
  - the *linguistic pattern* and the set of *enabling conditions* that guarantee its applicability.

  The conceptual anchor is a triggering word, while the enabling conditions represent constraints on the components of the linguistic pattern. The generation of concept nodes is based on the specialization of a set of predefined heuristics in the form of general linguistic patterns. The generation process is carried out by examining each example in an annotated corpus only once. Such linguistic patterns contain syntactic constraints which generate a concept node when a specialization occurs.
  For instance, in Figure 2.2, the linguistic pattern `<subject> passive-verb` is specialized into `<target> bombed` when examining the training example "The Parliament was bombed by the guerrillas". As a consequence, a concept node for the `<target>` slot of a bombing template is generated with `bombed` as trigger and constraining the slot-filler value to be subject (i.e., *S*) within the training example. The resulting set of concept nodes is proposed to the user, in order to be reviewed. This is due to the fact that AutoSlog makes no attempt to generalize examples, and, consequently, generates very specific slot-filler definitions (i.e., rules with low coverage).
  AutoSlog uses a predefined set of 13 linguistic patterns, and the information to be extracted can fall under one of the following categories:

– the *subject* of the sentence (e.g., "`<subject> passive verb`" for "<target> was bombed").
– the *direct object* (e.g., "`gerund <dobj>`" for "*bombing <target>*".
– a *noun phrase* (e.g., "`noun prep <np>`" for "*bomb against <target>*").

In general, the triggering word is a verb, but if the information to be extracted is a noun phrase, the triggering word also may be a noun.

CONCEPT NODE	
Name:	target-subject-passive-verb-bombed
Trigger:	bombed
Variable Slots:	(target (*S* 1))
Constraints:	(class phys-target *S*)
Constant Slots:	(type bombing)
Enabling Conditions:	((passive))

Fig. 2.2. A concept node induced by AutoSlog

In Figure 2.2, the complete definition of a sample concept node is showed. The *Name* slot is a concise, human readable description of the concept, and it is composed of the information to be extracted (i.e., the *target* of a terrorist attack), the linguistic pattern "`subject passive-verb`", and the triggering word *bombed.* The *Trigger* slot defines the triggering word, while the *Variable Slot* specifies the information to be extracted (in the example, the subject of the sentence). Finally, the sample concept requires that the subject is a physical target and that the verb is used at its passive form. It is easy to see that the above concept can be used to extract the target of the terrorist attack in the sentence "The Parliament was bombed by the guerrillas" but we need an `active-verb`-based concept to perform the same task for the sentence "The guerrillas bombed the Parliament".

- **PALKA**
  PALKA [83] is based on a candidate-elimination algorithm able to learn both single-slot and multi-slot rules. PALKA learns extraction patterns that are expressed as *Frame-Phrasal pattern structures* or, shortly, FP-structures. As shown in Figure 2.3, an FP-structure consists of a *meaning frame* and a *phrasal pattern.* Each slot in the meaning frame defines an item-to-be-extracted together with the semantic constraints associated to it (e.g., the target of a bombing event must be the type PHYSICAL-OBJECT). The phrasal pattern represents an ordered sequence of lexical entries and/or sematic categories. PALKA generalizes and specializes such semantic classes by using an ad-hoc semantic hierarchy until the resulting FP-structures cover all the initial specific ones. The use of the semantic

hierarchy allows PALKA to learn rules that are more general than AutoSlog's.

```
The Meaning Frame:
  (BOMBING
     agent:          ANIMATE
     target:         PHYSICAL-OBJECT
     instrument:     PHYSICAL-OBJECT
     effect:         STATE)

The Phrasal Pattern:
((PHYSICAL-OBJECT) was bombed by (TERRORIST-GROUP))

FP-structure = MeaningFrame + PhrasalPattern
  (BOMBING
     target:        PHYSICAL-OBJECT
     agent:         TERRORIST-GROUP
     pattern:( (target) was bombed by (agent) )
```

Fig. 2.3. Example of FP-structure

The FP-structure combines the meaning frame and the phrasal pattern by linking the slots of the former to the elements of the latter. Applying an FP-structure to a sentence represents a straightforward process: if the phrasal pattern matches the sentence, the FP-structure is activated, and then the corresponding meaning frame is used to actually extract the data.

- **CRYSTAL**
  CRYSTAL [134] generates *concept node* extraction patterns that are similar in nature, but significantly more expressive than the ones used by AutoSlog. CRYSTAL allows both *semantic* and *exact word* constraints in any component phrases. Furthermore, CRYSTAL also allows the use of multi-slot extraction patterns.
  In order to better illustrate the expressivity of the CRYSTAL concept node, we will consider a slightly modified version of the previously used target-sentence. The extraction pattern shown in Figure 2.4 can be used to extract both the *target* and the *name* of the perpetrator from the sentence "The Parliament building was bombed by Carlos".
  As we can see in Figure 2.4, the concept definition includes semantic constraints on both the subject and the prepositional phrase, and it also imposes exact word matching for the verb and the preposition. Furthermore, the "`Terms include:`" construct introduces an additional exact word matching for the subject of the sentence.

```
Concept type: BUILDING BOMBING
  Subtype: Known-Individual-Perpetrator
  Constraints:
    SUBJECT:
      Classes include: <Physical Target>
      Terms include: BUILDING
      Extract: target
    VERB:
      Root: BOMB
      Mode: passive
    PREPOSITIONAL PHRASE:
      Preposition: BY
      Classes include: <Person Name>
      Extract: perpetrator name
```

Fig. 2.4. Example of concept node generated by CRYSTAL

- **RAPIER**
  RAPIER [22] is the first relational learning system that we review. It uses a bottom-up compression algorithm in which rules are iteratively merged, instead of generalized, from training examples. RAPIER considers training examples as specific rules. At each iteration, two rules (specific or not) are selected to be compressed into a new one. Rules used in this process are discarded and the resulting rule is added to the set of possible ones.
  More in detail, the RAPIER system learns unbounded ELIZA-like patterns [151] that use limited syntactic information (e.g., the output part of a part-of-speech tagger) and semantic class information (e.g., hypernim links from WordNet [101]). In order to understand how RAPIER works, an example of target text, extracted information, and extracted pattern is presented in Figure 2.5. Here, it is possible to see that RAPIER extraction pattern consists of three distinct slots: the "Pre-filler pattern" and the "Post-filler pattern" play the role of left and right delimiters, while the "filler pattern" describes the structure of the information to be extracted.
  Each filler pattern consists of a (possibly empty) list of *pattern items* or *pattern lists*. The former, matches exactly one word/symbol from the document, while the latter specifies a maximum length N and matches 0 to N word/symbols from the document. The constraints imposed by pattern items/lists consist of exact match words, parts of speech, and semantic classes. For instance, in the example of Figure 2.5, the pre-filler and the post-filler patterns specify that information to be extracted is immediately preceded by the word "leading" and is immediately followed either by "firm" or by "company". The filler-pattern imposes constraints on the structure of the information to be extracted: it consists of at most two words that were labeled "nn" or "nns" by the POS tagger [17] (i.e., one

```
Telecommunications.  SOLARIS System    Information to be extracted:
Admin.  38-44K. Immediate need.        ---------------------------
                                       computer-science-job
Leading telecommunication firm in      title:    SOLARIS System Admin
need of an energetic individual to     salary:   38-44K
fill the following position:           area:     telecommunications

     AREA extraction pattern:

       Pre-filler pattern:   1) word:  leading
       Filler pattern:       1) list:  len:  2
                                tags:  [nn, nns]
       Post-filler pattern:  1) word:  [firm, company]
```

Fig. 2.5. RAPIER: sample text, desired output, and extraction pattern

or two singular or plural common nouns).

- **SRV**
  SRV [52] is an ILP (Indictive Logic Programming) system that generates first-order logic extraction patterns on the basis of attribute-value tests and the relational structure of the documents. SRV transforms the problem of learning IE rules into a classification problem: is a document fragment a possible slot value? The input of this system is a training set of documents, and a set of attributive and relational features related to tokens T (e.g., capitalized(T), next(T1; T2)) that control the generalization process. SRV uses a top-down covering algorithm to learn IE rules from positive and negative examples.
  In Figure 2.6, a rule learned by SRV to extract the course number from HTML pages containing class description, is showed.

```
1:    CourseNumber:-
2:        length( =2 ),
3:        every( in-title false ),
4:        some( ?A [] all_upper-case true ),
5:        some( ?B [] tripleton true )

Example:

<title> Course Information</title> <h1> CS 231 C++ Programming </h1>
```

Fig. 2.6. Example of SRV extraction pattern

This sample extraction pattern has the following meaning: the course number (line 1) consists of two tokens (line 2). No token between <title> and </title> is part of the course number (line 3). One of the tokens in the course number, say "`?A`", is all in upper-case (line 4), and the other token to be extracted, say "`?B`", consists of three characters (line 5).

- **WHISK**
  A more flexible system within the relational learning paradigm is WHISK. WHISK [133] generates extraction patterns for a wide variety of online information ranging from structured text (i.e., rigidly formatted) to free text. Following a different approach, WHISK represents documents as sequences of tokens and allows learning of both single-slot and multi-slot rules to extract slot values.
  The WHISK extraction patterns have two distinct components: one that describes the *context* that makes a phrase relevant, and one that specifies the exact *delimiters* of the phrase to be extracted. Depending of the structure of the target text, WHISK generates patterns that rely on either of the components (i.e., context-based patterns for free text, and delimiters-based patterns for structured text) or both on them (i.e., for documents that lay in between structured and free text).
  In Figure 2.7, a sample WHISK extraction task from online texts, is showed. Concretely, rules are represented as pairs <*pattern*, *output*>, in which *pattern* is meant to be matched by documents and *output* is required to be the output template when a match occurs. The pattern is a regular expression that represents possible slot fillers and their boundaries. For instance, the target text in Figure 2.7, is taken from an apartment rental domain that consists of ungrammatical constructs, which, without being rigidly formatted, obey some structuring rules that make them human understandable. The information to be extracted consists of a pairs <*NumberOfBedrooms*, *Price*>. The sample pattern showed in this figure 2.7 has the following meaning: ignore all characters in the text until a digit followed by "`BR`" string is found. Extract this digit and use it to fill the "*Bedrooms*" slot. Then, ignore again all the remaining characters ("`*`") until a "`$`" immediately followed by a number is reached. Finally, extract the number and use it to fill the "*Price*" slot.
  A more sophisticated version of the pattern could replace the "`BR`" string by the semantic class "<*Bedroom*>", which is defined as:

  $$< Bedroom >::= (brs||br||bds||bdrm||bd||bedroooms||bedrooom||bed)$$

  In practice, the semantic class "<Bedroom>" represents a placeholder for any of the abbreviations above.
  The rules so far described are learned in a top-down fashion from a training set of positive examples. An unusual selective sampling approach is used

```
Capitol Hill - 1 br twnhme.  fplc         RENTAL:  Bedrooms:    1
D/W W/D. Undrgrnd pkg incl   $675.                 Price:     675
3BR, upper flr on turn of ctry HOME.
incl gar.grt N. Hill loc $995             RENTAL:  Bedrooms:    3
(206) 999-9999 <br>                                Price:     995


   EXTRACTION PATTERN:  * (<Digit>) 'BR' * '$'' (<Number>)
```

Fig. 2.7. WHISK: sample text, desired output, and extraction pattern

by WHISK. Initially, a set of unannotated documents is randomly selected as training input out of those satisfying a set of key words. These documents are presented to the user who tags the slot-fillers. WHISK starts by learning a rule from the most general pattern (e.g., '*(*)*' for single-slot rules). The growth of the rule proceeds one slot at a time. This is done by adding tokens just within the slot-filler boundaries as well as outside them. The growth of a rule continues until it covers at least the training set. After a rule set has been created, a new set of unannotated documents can be selected as a new training input from those satisfying the rule set. Although WHISK is the most flexible state-of-the-art approach, it cannot generalize on semantics when learning from free text, as CRYSTAL, PALKA, SRV and RAPIER do. Another limitation of WHISK is that no negative constraints can be learned.

Towards Unsupervised Approaches

The main bottleneck of the supervised pattern discovery methods is the cost of the preparation of training data. Many of the surveyed systems need a large amount of annotated document for a particular extraction task, and hence the cost for manual annotation cannot be ignored. This also leads to the lack of portability of an IE system: one needs to get the annotated data for each task.

Therefore, studies on information extraction have frequently focused on these drawbacks by requiring a lower degree of supervision. In the following, we first review a system representing an evolution of AutoSlog, and then, some approaches all related to certain form of learning known as *bootstrapping*.

- **AutoSlog-TS**
  AutoSlog-TS by Riloff [120] is a new version of AutoSlog where the user only has to annotate documents containing text as relevant or non-relevant before learning. The strategy of AutoSlog-TS consists of two stages. In the first, it applies the heuristic-driven specialization used by AutoSlog in order to generate all possible rules (concept nodes, see Figure 2.2) with the relevant documents. This is done by matching a set of general linguistic patterns against the previously parsed sentences of the relevant documents.

In the second stage, a relevance rate is computed for each one of the resulting rules as the conditional probability that a text is relevant given that it activates the particular rule. The relevance formula is the following:

$$Pr(relevant\_text|text\_contains\_rule_i) = \frac{rel\_freq_i}{total\_freq_i}$$

where rel_freq_i is the number of matches of $rule_i$ found in the relevant documents, and $total\_freq_i$ is the total number of matches of $rule_i$ found in the whole set of documents. Finally, each rule is ranked according to the formula:

$$\begin{cases} relevance\_rate(rule_i) * log_2(freq_i) & \text{if } relevance\_rate(rule_i) > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

and the n best ranked rules (n according to the user criteria) are selected. Riloff assumes that the corpus is 50% relevant and, consequently, when the relevance rate is lower or equal to 0.5, the rule is negatively correlated with the domain.
The author presented a comparison between AutoSlog and AutoSlog-TS, related to the learning of single-slot rules to extract three slots in terrorist domain (*perpetrator*, *victim*, and *target*). The main conclusion was that AutoSlog-TS can extract relevant information with comparable performance to AutoSlog's, but requiring significantly less supervision and being more effective at reducing spurious extractions. However, the relevance rate formula tends to rank many useful rules at the bottom and to rank high frequency rules at the top. The author justifies that behavior with the necessity of a better ranking function.

- **Bootstrapping approaches**
Some research groups have been focusing on the use of a certain form of learning known as *bootstrapping*. Bootstrapping is a general framework for improving a learner using unlabeled data [81]. Typically, bootstrapping is an iterative process where labels for the unlabeled data are estimated at each round in the process, and the labels are then incorporated as training data into the learner. All the approaches exploiting bootstrapping are based on the use of a set of either seed examples or seed patterns from which they learn some context conditions that then enable them to hypothesize new positive examples, from which they learn new context conditions, and so on. In general, all the information extraction methods following such approach use a bottom-up covering algorithm to learn rules. In the following we review some of these bootstrap-based techniques.
DIPRE (*Dual Iterative Pattern Expansion*) [18] is a system for acquiring patterns, similar to co-training [14], which is able to extract binary relations from web documents. Very simple patterns are learned from a set

of seed word pairs that fulfil the target relation (for example, Company - Location). The seed word pairs are used to search web pages for text fragments where one word appears very close to the other. In this case, a pattern is created which expresses the fact that both semantic categories are separated by the same lexical items that separate the example seed words in the text fragment found. A pattern is composed by five string fields: *prefix*, *category1*, *middle*, *category2*, *suffix*. A text fragment matches the pattern if it can be split to match each field. For instance, to learn the relation (Author, Book Title) from web pages, DIPRE learned the pattern "`<LI><B>title</B> by author`", where the text preceding the title is the *prefix*, the text between the title and the author is the *middle* and the *suffix* consists of the text following the author. The set of patterns obtained from the example relations are used to find new pairs of related words by matching the patterns with the present set of web pages and the process is repeated.

Mutual bootstrapping is one form of co-training used in Riloff and Jones [121] for lexical discovery. Lexicons and extraction patterns are used as two separate features. Given a handful of lexical entries as initial data, patterns are discovered that extract the initial lexicon. The extracted patterns are ranked and the most reliable are used to extract more lexical items. The assumption of using *mutual bootstrapping* is the duality that a good pattern can find a good lexicon and a good lexicon can find a good pattern. Thus, a set of lexicons would find the extraction patterns that can reliably extract them, and, in turn, the extraction patterns can find another set of lexicons, while keeping the relevancy of the lexicons. A strong limitation of mutual bootstrapping is that a minor error can cause a large amount of errors during the following iteration. A touching up process was introduced by *meta-bootstrapping*. In meta-bootstrapping, each iteration takes only the five best noun phrases for adding to the extracted lexicons, thus trying to be conservative rather than taking all the lexical entries that are extracted by the patterns.

ExDISCO [157, 158] is another interesting extraction strategy with mutual bootstrapping. Here, extraction patterns in the form of subject-verb-object (SVO) are learned from an initial set of SVO patterns manually built. The application of these initial patterns in a text indicates that the text is suitable for extracting a target event or a part of it. The motivation behind the mutual bootstrapping of ExDISCO is the circularity where the presence of the relevant documents indicates good patterns and good patterns can find relevant documents.

ExDISCO requires only an unannotated corpus and a handful of seed patterns. First the document set is divided into a *relevant document set* that contains at least one instance of patterns and *non-relevant document set* that do not contain any seed patterns. The candidate patterns are generated from the clauses in the documents and ranked in correlation with the relevant documents. The score of pattern pt is calculated by:

$$score(pt) = \frac{doc_R(pt)}{doc(pt)} \cdot \log(doc_R(pt))$$

where $doc_R(pt)$ is the number of relevant documents that contain pt. ExDISCO adds the highest pattern to the pattern set that initially contains only seed patterns. Then, ExDISCO re-ranks each document using the newly obtained pattern set. At the $(i+1)-th$ iteration, the score of each document d is calculated by:

$$score^{i+1}(d) = max\left(score^i(d), \frac{1}{|DOC(PT_d)|} \cdot \sum_{d \in DOC(PT_d)} score^i(d)\right)$$

where PT_d is a set of patterns that match document d and $DOC(PT_d)$ is a set of documents that all patterns in PT_d match. This score calculates the sum of the normalized scores of the documents which contain all the patterns that now are found in the document in question. Thus, having all the documents updated score, ExDISCO again spits the entire set of documents into *relevant* and *non-relevant* and keeps iterating.
SNOWBALL [2] is a system able to learn local relations between entities. SNOWBALL approach is based on DIPRE algorithm (see above in this section). The main algorithm of SNOWBALL, first, induces and ranks the extraction patterns given a handful of initial relation instances; a set of pairs of location and organization. The algorithm applies the patterns onto the source text and tries to find other relations that the patterns can cover. From the newly found relations, the patterns are re-ranked. The metrics of the patterns and relations used for each iteration are important since a tiny error would extract more erroneous patterns/relations in the later stages. The confidence of a pattern is calculated by similar method to AutoSlog-TS [120], where the score is a multiplication of the relevance of the pattern and the logarithm of the frequency of the pattern (see equation (2.1)). Then, the confidence of each relation is calculated by the probability that all the patterns that extract the relation were triggered incorrectly.

2.4 Stochastic Approaches

Although rule learning techniques have been the most common ones used for IE, several approaches explore the use of well-known statistical machine learning methods which have not been previously applied to this area. These methods include *Hidden Markov Models* (HMMs), *Maximum Entropy Models* (MEMs), and *Conditional Random Fields* (CRFs). This section is devoted to a description of these approaches when applied to information extraction tasks.

2.4.1 Hidden Markov Models

Hidden Markov Models (HMMs) are a type of probabilistic finite state machine, and a well-developed probabilistic tool for modeling sequences of observations. They have been applied with significant success to many language-related tasks, including part-of-speech tagging [87], and speech recognition [116]. Instead, relatively recent is the exploitation of the HMMs in the IE scenario. Several are the reasons for which the HMMs are used in the information extraction. First of all, they have a strong statistical foundations, and showed to be well-suited to natural language domains. Moreover, the HMMs are able to handle new data robustly, and they are computational efficient to develop. However, as in the case of many machine learning approaches, large amounts of training data are required to learn a model that generalizes well and has high accuracy.

Hidden Markov Models for Information Extraction

A HMM is a finite state automaton with stochastic state transitions and symbol emissions [116]. The automaton models a probabilistic generative process whereby a sequence of symbols is produced by starting at a designed *start* state, transitioning to a new state, emitting a symbol selected by that state, transitioning again, emitting another symbol, and so until a designed *final* state is reached. Associated with each set of states $S = \{s_1, \ldots, s_n\}$, is a probability distribution over the symbols in the emission vocabulary $V = \{w_1, \ldots, w_m\}$. The probability that state s_j will emit the vocabulary item w is indicated with $P(w|s_j)$. Similarly, associated with each state is a distribution over its set of outgoing transitions. The probability of moving from the state s_i to state s_j is indicated with $P(s_j|s_i)$. Model transition and emission probabilities can be learned from training data (*training phase* of a HMM). Training data consists of several sequences O of observed emissions, one of which would be written as $\{o_1, \ldots, o_k\}$. In practice, $\{o_1, \ldots, o_k\}$ represents a generic document from which extracting the interesting information.

In order to practically understand the way a HMM may be used for information extraction purpose, consider the next example, in which the objective consists in extracting information from research paper headers. A model that can be used in this scenario is the following: each state is associated with a class that must be extracted, such as title, author or affiliation. In order to label a new header with classes, the words from the header are considered as observations, and the most-likely state sequence is recovered. The state that produces each word is the class tag for that word. In the Figure 2.8, an example of HMM annotated with class labels and transition probabilities is showed.

In general, given a HMM model M and all its parameters, information extraction is performed by determining the sequence of states that is most likely to have generated the entire document, and extracting the symbols that

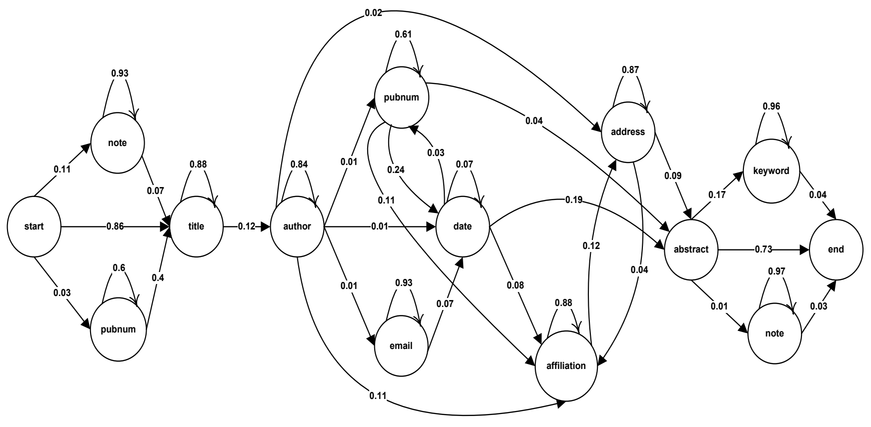


Fig. 2.8. Example of a HMM in a typical IE scenario

are associated with such states. This phase is also known as *testing phase* of a HMM. To perform extraction we therefore require an algorithm for finding the most likely state sequence given a model M and a sequence of symbols.

Intuitively, the probability that an output sequence $\mathbf{o} = \{o_1, \ldots, o_k\}$ (i.e., a string) being emitted by a HMM model M is computed as a sum over all possible paths by:

$$P(\mathbf{o}|M) = \sum_{s_1,\ldots,s_k \in S^k} \prod_{i=1}^{k+1} P(s_i|s_{i-1})P(o_i|s_i)$$

where s_0 and s_{k+1} are restricted to be the initial and final states respectively, and o_{k+1} is an end-of-string token. The *Forward algorithm* can be used to calculate this probability [116]. The state sequence $S^*(\mathbf{o}|M)$ that has the highest probability of having generated an observation sequence $\mathbf{o}$ can be stated as:

$$S^*(\mathbf{o}|M) = \arg \max_{s_1,\ldots,s_k \in S^k} \prod_{i=1}^{k+1} P(s_i|s_{i-1})P(o_i|s_i)$$

Although a naïve approach to find the most likely sequence would take $O(k^n)$, a dynamic programming solution, called *Viterbi algorithm* [148], solves this problem in just $O(kn^2)$ time.

In the above proposed example, a single HMM is used to extract a set of fields from quite-structured texts (e.g. paper headers), taking into account field sequence. The fields are close to each other, and thus this approach is also referred as "*dense extraction*" task. A different IE task could consist in extracting relevant data from documents containing a lot of irrelevant text and thus referred as *"sparse extraction"* task. Some works (e.g., [53, 54]) propose a possible HMM model able to deal with the sparse extraction task.

The training of a HMM has two phases: the choice of its structure (the *topology*), and the learning of transition/emission probabilities.

Usually, building a HHM it implies the necessity of an a priori notion of its topology (the number of states and the transitions between the states). This is one of main drawbacks of HMMs, although recently techniques were developed to automatically identify a good task-specific topology [54]. In general, it is difficult to get the optimal number of states in the HMM. In literature, several are the possible topologies. *Fully-connected model*, *naïve model*, and *left-right model* are only some possible structures. However, once the model is chosen, it is possible to subsequently refine the topology of a HMM. To this purpose, some techniques exist:

- The *Neighbor-Merging* technique combines all states that share a transition and have the same class label. For instance, by newly referring to the header extraction task, the sequence of adjacent title states from a single header are merged into a single title state. As multiple neighbor states with the same class label are merged into one, a self-transition loop is introduced, whose probability represents the expected state duration for that class.
- The *V-merging* technique merges any two states that have the same label and share transitions from or to a common state.
- The *Bayesian model merging* [137] technique seeks to find the model structure that maximizes the probability of the model M given some training data D. This is achieved by iteratively merging states until an optimal tradeoff between fit to the data and model size has been reached.

Once a model structure has been selected, the transition and emission probabilities need to be estimated from training data. When the training sequences are sufficiently labeled so as to be associated with a unique path of states, the probabilities can be calculated straightforwardly with ratios of counts (*maximum likelihood*) or smoothed ratio of counts (*maximum a posteriori*). Accordingly, the probability of making a transition from a state s_i to state s_j, can be calculated as follows:

$$P(s_j|s_i) = \frac{\text{Number of transition from state i to state j}}{\text{Total number of transitions out of state i}}$$

The emission probabilities are computed similarly. The probability of emitting symbol w in state s_j can be written as:

$$P(w|s_j) = \frac{\text{Number of times the w symbol emitted at state j}}{\text{Total number of symbols emitted at state j}}$$

However, the above formula for emission probabilities needs to be refined when the training data are insufficient. Often, during testing, it is possible to encounter words that have not been seen during training. When the emission vocabulary is large with respect to the number of training examples, maximum likelihood estimation of emission probabilities will lead to poor estimates, with many words inappropriately having zero probability. Hence, assigning a correct probability to the unknown words is important. The use of a well-chosen prior in conjunction with maximum a posteriori or Bayes optimal estimation will prevent zero-probability estimates and improve estimation overall.

For instance, Bayes optimal parameter estimation in conjunction with a uniform Dirichlet prior results in the widely used *Laplace smoothing*, in which the count of every word in the vocabulary is incremented by a single extra "priming" occurrence. This is also known as *additive smoothing*. An alternative smoothing technique that performs better when the number of zero-count words varies widely from state to state is *absolute discounting*. This method subtracts a fixed discount, $0 < d < 1$ from all words with count greater than zero. The resulting available probability mass is then distributed over all words that have zero count according to some prior distribution.

HMM-based approaches

Several recent researches have demonstrated the effectiveness of hidden Markov models for information extraction. HMMs can be applied successfully to subdomains of information extraction as *(i)* the task of segmenting structured data sources provided as continuous, unordered, and non delimited text [15, 1] (e.g., addresses or bibliographic records), *(ii)* the task of recovering the sequence of a set of entries occurring in close proximity (e.g., headers of research papers) [127], and *(iii)* the "template-filling" task in which the objective is to extract relevant phrases from document containing much irrelevant text [53, 54]. In many cases, the accuracy of HMMs applied to these tasks is state-of-the-art and often significantly better than alternative learning approaches.

- **Estimating parameters with shrinkage**
  Freitag and McCallum [53] assume that for every document in a corpus there is a corresponding relational record (template), each field of which is either empty or is filled with a fragment of text from the document. For instance, an electronic seminar announcement might contain the title of the talk, the name of the speaker, the starting time, etc. In the proposed methodology, a separate HMM is constructed by hand for each target slot (e.g., seminar speaker) to be extracted. Each model contains two types of states, *target states* that produce the tokens to extract and *non-target states*. The structure of each HMM focuses on modeling the immediate prefix, suffix, and internal structure of each slot. For each HMM, both the state transition and word emission probabilities are learned from labeled data. However, they integrate a statistical technique called *shrinkage* in order to be able to learn more robust HMM emission probabilities when dealing with data-sparseness in the training data (large emission vocabulary with respect to the number of training examples). More in detail,

parameter estimates from sparse states in a complex model are "shrinked" towards estimates from related states of simpler models where more training data is available for each state (because the number of states is lower). All target states are considered as related, as are all non-target states. A weighted average learned through *Expectation-Maximization* [44] is used to combine the estimates of different models. The smoothed, shrinkage-based probability of state s emitting word w is:

$$\lambda_1 P(w|s) + \lambda_2 P(w|a(s)) + \lambda_3 \left(\frac{1}{K}\right)$$

where the last term represents the uniform distribution, $a(s)$ is the "parent" state of s (a state combining all target states if s is a target state, a state combining all non-target states otherwise) in the shrinkage hierarchy, and $\lambda_1 + \lambda_2 + \lambda_3 = 1$.
In fact, the type of shrinkage used, which averages among different HMM states (the ones with poor data versus the data-rich ones), is the one known in speech recognition as *deleted interpolation.* The method has been evaluated on the domains of on-line seminar announcements and newswire articles on corporate acquisitions, in which relevant data must be recovered from documents containing a lot of irrelevant text.

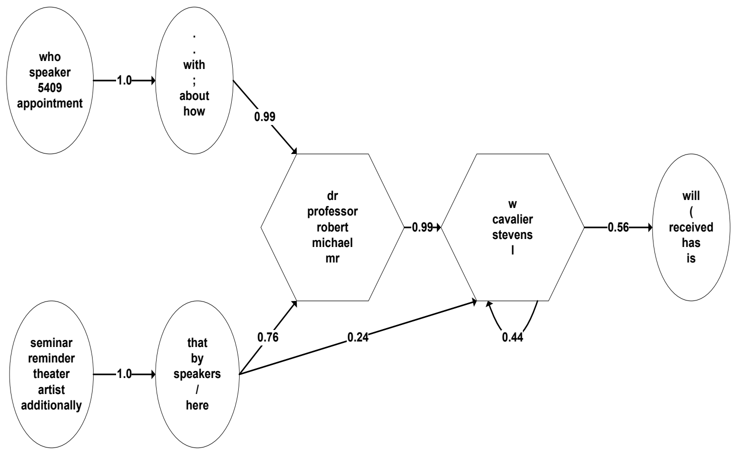


Fig. 2.9. Part of the HMM structure for extracting the speaker field

Figure 2.9 shows part of the structure of an HMM for extracting the speaker field in the on-line seminar announcement domain. The elliptical nodes represent the prefix/suffix states of the field to be extracted, whereas the polygonal nodes represent the field states themselves. In both types of nodes, the top 5 most probable tokens to be emitted by that state are shown. Only those transition probabilities greater than 0.1 are depicted. The authors claim better results than the SRV system (see Section 2.3.2), although they need the a priori definition of the topology of the model.

- **Learning the structure using stochastic optimization**
  In an extension of the previous approach, Freitag and McCallum [54] tackle again the sparse extraction task, but this time the work focuses on robustly learning a HMM structure for each target slot from limited specific training data.
  For learning a suitable HMM structure, non-target states are further differentiated as either *prefix* or *suffix* (preceding resp. following a target phrase) or *background* states (anything else). The most simple HMM fitting this structure has four states (one of each kind) and considers exactly one prefix + suffix around each target state[1]. Starting from this model, a stochastic hill-climbing process is performed in the space of possible structures. Related models are generated by *lengthening* a prefix, suffix, or target string (adding a new state of the same kind that must be traversed before the model can proceed to the next kind of state), by *splitting* a prefix/suffix/target string (creating a duplicate where the first and last states of the duplicated prefix/suffix/target have the same connectivity as in the original), or *adding* a background state. Model variations are evaluated on a hold-out set or via 3-folds cross-validation. It is worth noticing that, in this approach, the parameter estimation is performed as well as in their previous work [53].
  In order to evaluate the goodness of this approach, the score used is F_1 (the harmonic mean of precision and recall), evaluated on the training data from the same two domains as their previous work [53], along with the semi-structured domains of job announcements and *Call for paper* announcements. Training data must be labeled. Experimental results show a higher accuracy than the one achieved by their previous approach, as well as the ones from SRV and RAPIER systems (see Section 2.3.2).
- **Learning structure and parameters from training data**
  In contrast to the previous approach, Seymore et al. [127] presents a method for both learning the HMM's topology and training the HMM (estimating the probabilities of both the transitions between the states and the emission of class-specific words from each state) from the data. The approach uses a single HMM to extract a set of fields from quite-structured texts (e.g. computer science research paper headers), taking into account field sequence. The fields are close to each other ("dense extraction"). While the selection of the model structure needs, in order to

[1] The background state is connected to itself and to the prefix state which is in turn connected to the target state; the target state is connected to itself and to the suffix state which is connected to the background state.

be accomplished, data labeled with information about the target-slot to be extracted, the HMM parameters can be estimated either from labeled data (via maximum likelihood estimates) or from unlabeled data (using the widely known Baum-Welch training algorithm [8]). A good step towards portability is the introduction of the concept of *distantly-labeled data* (labeled data from another domain whose labels partially overlap those from the target domain), whose use improves classification accuracy. On the other hand, a clear drawback is the need of large amounts of training data in order to maximize accuracy.

- **DATAMOLD**
  An approach strictly related to the work of Seymore et al. [127] is *DATAMOLD* [15]. The main goal of this system is to automatically segment structured data sources provided as continuous text (addresses principally) into known elements or classes (House No., City, State, Zip code etc.). The used model is a single HMM in which each state is associated with one element, and hence the state that produces each word is the element tag for that word. Several are the improvements introduced respect to the approach in [127], regarding essentially the structure of the model, the introduction of a taxonomy for the symbols, and the exploitation of information in external databases to optimize the model.
  The first attempt of *DATAMOLD* to improve the robustness of HMMs consists in overtaking too simple HMM topologies as the classical naïve structure. In the naïve structure, as many states (completely connected between them) as the number of elements are considered. This simple model is able to capture the ordering relationship amongst elements, but, because it has just one state per element, it ignores any sequential relationship amongst words in the same element. For instance, most road names end with words like "*Road*", "*Street*" or "*Avenue*". Treating an element as a single state does not capture this structure. Also, for country names like "*New Zealand*", both the "*New*" and "*Zealand*" will be outputs of the same state. The other problem is that it learns only a limited kind of distribution on the number of words per element. To face these drawbacks, a nested HMM (*inner HMM*) for each element is proposed in order to capture its internal structure. The *outer HMM* captures the sequencing relationship amongst elements, treating each inner HMM as single state.
  The "overall" HMM is learnt in a hierarchical manner in two stages. In the first stage the outer HMM is learnt. In this phase, the training data is treated as a sequence of elements ignoring all details concerning the length of each element and the words it contains. In the second stage, the inner HMMs are learnt. Here, the training data for each element is the sequence of all distinct tokens (word, delimiter, digit) in the element.
  An element typically has a variable number of tokens. For instance, city-names most frequently have one but sometimes two or more tokens. Having a inner HMM as in Figure 2.10, such a variability can be handled by choosing the path of length one, two or more, respectively.

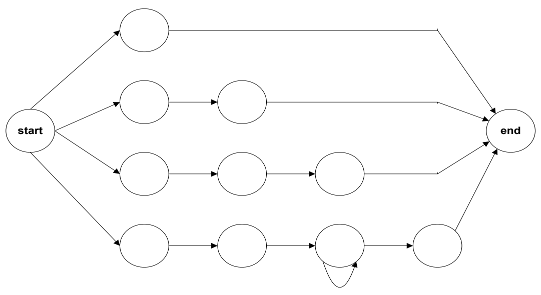


Fig. 2.10. An example of inner HMM in *DATAMOLD*

In order to deal with sequences containing unknown tokens, a non trivial issue in the training phase, is the choice of the symbols in the dictionaries of the states. The usual approach typically consists in treating each word, number or delimiter in the text as a token. However, in an address such as "*145 Sunset Blvd Los Angeles CA 90027*", it is interesting to recognize "*145*" as a number and not as a word, or recognize "*90027*" as a five-digit number in order to assign it the Zip code as element. To automate this choice, *DATAMOLD* uses a hierarchical arrangement of the features. An example of taxonomy is showed in Figure 2.11 where at the top most level there is no distinction amongst symbols; at the next level they are divided into "Numbers", "Words", and "Delimiters"; and so on. The right level of generalization of symbols in the training data can affect the selection of the correct path in the HMM, and hence it is a critical task. The level of generalization is chosen in a bottom-up manner with a process similar to the way the decision trees are pruned using cross-validation to avoid overfitting.

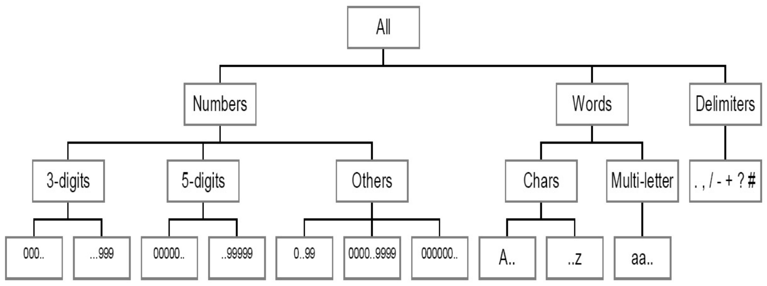


Fig. 2.11. An example taxonomy on symbols used in *DATAMOLD*

Finally, in *DATAMOLD* a further possible optimization of the model is proposed. Essentially, in order to make more accurate the HMM, *DATAMOLD* exploits additional external knowledge when it is available. This knowledge could be, for instance, in the form of a database of richer semantic relationship amongst symbols of different elements.
For the address data, a hierarchical database of countries, the state in each country, and cities in each state, can be available. In order to understand the way this database can be exploited, let consider the usual case where a city name is followed by either a state or a country name. Solving this ambiguity it becomes easy: it suffices to verify in the database if a certain state/country has a city with such a name.
In practice, such information constraints the combination of values that are allowed in different elements, and could be useful in order to find the right assignment of symbols to states.
To incorporate dependency information in HMMs, a variant of the Viterbi algorithm [148] is implemented. The original formulation is modified in order to restrict the exploration of paths that are invalid given the database of semantic relationships amongst symbols of different elements.

- **CRAM**
An unsupervised approach to text segmentation is introduced in [1]. The basic idea here is to exploit *reference relations* for building segmentation models. The notion of reference relation denotes a collection of structured tuples, that are specific to a domain of interest and exemplify clean records for that domain. However, it is not possible to directly adapt existing supervised approaches (e.g., *DATAMOLD*) to the "learn from reference table only" scenario. First, because from the reference relations is unknown the order in which the class values (or attribute values, since a relation context is considered) would be observed in test input sequences. Second, because data in reference tables is usually clean, whereas input data typically is dirty.
The proposed segmentation system *CRAM* (*Combination of Robust Attribute Models*) consists of a two-steps process. Let assume R a reference relation with an attribute schema $\{A_1, \ldots, A_n\}$. Each column of R is considered as a dictionary of basic values for the corresponding attribute. In the first step, a *preprocessing* phase is performed for building an attribute recognition model (ARM) for each attribute of the reference relation schema. The generic ARM_i is a HMM that allows the evaluation of the probability with which a subsequence of tokens in an input string belongs to the domain of the corresponding schema attribute A_i. Improvements in the instantiation of $ARMs$ are proposed both in the building of structure and in the estimation of parameters. The topology of each ARM, since it is trained on "reference data", must assure less sensitivity respect to the order of tokens in an attribute value. To this purpose, these tokens, and therefore the states of an ARM, are classified into three categories: *beginning*, *middle*, and *trailing* positions/states. The parameter estimation

phase of each ARM is optimized by exploiting a feature hierarchy (similar to that in $DATAMOLD$) helping to recognize tokens that have not been encountered previously. Once the ARM for each attribute value is built, all these $ARMs$ can be exploited to determine the best segmentation of an input string at the second *segmentation* step. This involves, first, to learn the total order of attributes from a batch of input strings and, subsequently, to segment the individual input strings with respect to the detected attribute order. More specifically, the identification of a total attribute order requires the previous computation of pairwise precedence probabilities. These are probabilistic estimates of precedences between all pairs of attributes, that are provided by their corresponding $ARMs$. A total ordering among all of the attributes is hence discovered by choosing the best sequence of attributes, i.e. the sequence that maximizes the product of precedence probabilities of consecutive attributes with respect to the given order. Finally, an exhaustive search is employed to determine the best segmentation of an input string s into n token subsequences $s_1, \ldots, s_n$, such that the reconciliation of each s_i with the corresponding schema attribute A_{si} maximizes the overall reconciliation quality $\prod_{i=1}^{n}\{ARM_{si}(s_i)\}$ among all possible segmentations.
Notice that the exploitation of reference tables is a natural way of automatically building training sets for the text segmentation problem. Therefore, although declared as an unsupervised approach, this technique suffers from two general weaknesses, that are inherent of supervised methods. Foremost, a reference relation may not exist for a particular applicative scenario. Also, whenever the overall number of tuples involved is not sufficiently large, the columns of the employed relations may not be adequately rich dictionaries of basic domain tokens. This would affect the construction of $ARMs$ and, hence, the overall segmentation effectiveness.

- **Other approaches**
  Skounakis et al. [130] use *hierarchical hidden Markov models* (HHMMs) [51] for IE. HHMMs combine several levels of states to describe a sequence at different granularity levels. A two-levels HHMM is used. The top level models phrase segments (noun, verb, and prepositional phrases) provided by a shallow parser, the lower level models individual words (including their POS tags) within a phrase. The Viterbi, Forward, and Backward algorithms are adapted to ensure that the embedded word model reaches the end state exactly at the end of a each phrase and to ensure the typing of the phrase model (each state has a type that corresponds to the type of the phrase segment it emits).
  Context hierarchical hidden Markov models (CHHMMs) are an extended variant that incorporate additional sentence structure information in each phrase. The word model is extended to consider the left and right neighbor of each word, generating a sequence of overlapping *trigrams*. To reduce the number of possible observations, individual features (words and tags) are combined under the assumption of conditional independence. Evalua-

tion shows superior results for hierarchical models, especially CHHMMs, compared with flat HMMs.

2.4.2 Maximum Entropy Models

Maximum Entropy models (MEMs) are introduced to integrate heterogeneous and possibly overlapping information from different sources. This information is represented in the model as binary or real valued feature functions defined over the combined space of data points and labels. Given a training set of labeled examples, the set of possible models is constrained to be the models where the expected value of each feature equals the average value of that feature in the training data. Out of this set of possible models, the principle of Maximum Entropy dictates that the model is chosen that has the highest entropy. The intuition behind this is that this model makes no assumptions beyond those dictated by the empirical data, since it is the most uniform model possible given the constraints derived from that data.

More formally, assume to model a distribution $P \in \mathcal{P}$ ($\mathcal{P}$ is the space of all conditional probability distributions) over a set of events X consisting of labeled observations drawn from the event space $E = O \times L$, where O is the observation space and L the label space. Moreover, suppose that a d-dimensional feature function $f : O \times L \to \mathbb{R}^d$ that characterizes these events is given. Taking $\tilde{P}(\cdot)$ to denote the empirical distribution of the operand in the training data, and $P(\cdot)$ to denote the probability of the operand assigned by the model, the above cited constraint can be reformulated as follows:

$$\sum_{o,l} \tilde{P}(o,l) f(o,l) = \sum_{o,l} \tilde{P}(o) P(l|o) f(o,l) \tag{2.2}$$

From the subset of models satisfying Equation (2.2), Maximum Entropy requires to choose the model that is most uniform, using conditional entropy as definition of uniformity:

$$H_P(L|O) = -\sum_{o,l} \tilde{P}(o) P(l|o) \log P(l|o)$$

Thus, the problem of finding the Maximum Entropy model, given the empirical distribution $\tilde{P}(L,O)$ is a constrained optimization problem. Defining C to be the set of constrained models satisfying Equation (2.2), this problem can be formulated as:

$$\hat{P} = \arg\max_{P \in C} H_P(L|O) \tag{2.3}$$

Introducing *Lagrangian multipliers* for each of the constraints imposed by the features, Berger et al. [10] show that the model $\hat{P}$ is the model of the parametric form P_λ below:

$$P_\lambda(l|o) = \frac{1}{Z_\lambda(o)} \exp\left(\sum_i \lambda_i f_i(o,l)\right) \tag{2.4}$$

where $\{\lambda_1 \ldots \lambda_d\}$ are the parameters to be learned, and $Z_\lambda(o)$ is a normalizing factor determined by the requirement that $\sum_l P_\lambda(l|o) = 1$.

Essentially, the parameter estimation problem for the maximum entropy models consists in finding the λ parameters that satisfy the constrained optimization problem posed in Equation (2.3). As the entropy is concave in the parameter space, there is a single global optimum, and the different estimation methods proposed are all iterative (hill-climbing) methods that differ only in convergence speed and ultimate proximity to the global maximum, not in the quality of the solution found. However, as the estimation problem can be quite daunting computationally due to the high number of free parameters, the rate of convergence can be of great practical value.

Traditionally, parameter estimation is done using an implementation of *Iterative Scaling*, either *Generalized Iterative Scaling* (GIS) [40] or *Improved Iterative Scaling* (IIS) [43]. Both are iterative methods that scale the probability distribution by a factor proportional to the ratio of the estimated values of the features under the current estimated parameters to the estimated value of the features under the empirical distribution.

Maximum Entropy Markov Models for Information Extraction

As previously discussed (see Section 2.4.1), in the HMM traditional approach the emission probabilities are typically represented as a multinomial distribution over a discrete, finite vocabulary of words. However, in some extraction tasks, this approach could be reductive. As a matter of fact, many tasks would benefit from a richer representation of observations in terms of many overlapping features, such as capitalization, word endings, part of speech, formatting and so on. As an example, consider the task of segmenting the questions and the answers of a FAQ list. The features that are indicative of the segmentation are not just the individual words themselves, but features such as the line length, indentation or the total amount of whitespace or grammatical features. Moreover, in the HMM traditional approach the parameters are set in order to maximize the likelihood of the observation sequence. However, in most text applications (including the FAQ example above), the task is to predict the state sequence *given* the observation sequence. In other words, the traditional approach in a inappropriate manner uses a generative *joint* model in order to solve a *conditional* problem in which the observations are given. In order to address such concerns a new model called *Maximum Entropy Markov Models* (MEMMs) [96] is introduced.

As viewed in Section 2.4.1, traditional hidden Markov model is given by a finite set of states S, a set of possible observations O, and two probability distributions: the transition probability gives the probability of going to a new state $s \in S$ given the current state $s' \in S$ ($P(s'|s)$), and the emission probability gives the probability of an observation $o \in O$ based on the current state $s \in S$ ($P(o|s)$). In the HMMs, the observation is conditionally independent of all other variables given the state it belongs to. In contrast, the MEMMs are governed by a single probability distribution $P_{s'}(s|o) = P(s|s', o)$ that provides the probability of the current state s given the previous state s', and the current observation o.

The use of state-observation transition functions rather than the separate transition and observation functions in HMMs, it allows to model transitions $P_{s'}(s|o)$ in terms of multiple, non-independent features of observations (e.g., POS tags, capitalizations, positions in the document, etc.) from which they could benefit. To this purpose, exponential models fitted by maximum entropy can be exploited.

The above introduction on MEMs recalled that maximum entropy is a framework for estimating probability distribution from data. It is based on the principle that the best model for the data is the one that is consistent with certain constraints derived from training data, and is that which is closest to the uniform distribution or, in other words, that with highest entropy. Essentially, such constraints indicate some characteristic of the training data that should be also present in the learned distribution, and in the MEMMs are based on binary features. Examples of such features might be "the observation is the word *apple*" or "the observation is a capitalized word" or, if the observations are whole lines of text at time, "the observation is a line of text that has two noun phrases".

If the applied constraints request that the expected value of each feature in the learned distribution must be same as its average on the training observation $\{o_1, \ldots, o_m\}$ (corresponding to the state sequence $\{s_1, \ldots, s_m\}$), then the probability distribution $P_{s'}(s|o)$ has maximum entropy and an exponential form as in Equation (2.4).

The *Generalized Iterative Scaling* algorithm is used to train the λ parameters of the model, and the most probable state sequence, given an observation sequence, is found using a variation of Viterbi algorithm adjusted for MEMMs. Instead, a variation of the Baum-Welch algorithm can be used to estimate missing states during training, so the model can be trained from partially labeled or even unlabeled documents.

Tested on a text segmentation task, MEMM has showed to perform significantly better than both classical HMMs and a stateless maximum entropy model. However, MEMMs present also some drawbacks.

The main weakness of MEMMs is the *label bias problem* [88]: the probability mass arriving at a state must be distributed among the successor states, thus outgoing transitions from a state compete only against each other, not against other transitions. This results in a bias in favor of states with fewer outgoing transitions. The *Conditional Random Fields* (CRFs) [88, 97, 95] overcome this problem.

A classification-based approach

Chieu and Ng [29] make use of the maximum entropy framework, like McCallum et al. [96], but instead of basing their approach on Markov models, they use a classification-based approach. A set of features are defined for each domain of application, from which the probability distribution is estimated that both satisfies the constraints between features and observations in the training corpus and makes as few additional assumptions as possible (according to the maximum entropy principle). They develop two techniques, one for single-slot information extraction on semi-structured domains and the other for multi-slot extraction on free text. The first one is applied to the Seminar Announcements domain. A trained classifier distributes each word into one of the possible slots to be filled (classes). The more complex multi-slot extraction task is applied to the Management Succession domain (using the same training and test data as *WHISK*). A series of classifiers is used to identify relations between slot fillers within the same template. The parameters of the model are estimated using the GIS algorithm.

2.4.3 Conditional Random Fields

Conditional random fields (CRFs) [88] are another type of conditional-probability finite state model for labeling and segmenting sequential data. CRFs represent a sequence modeling framework that has all the advantages of MEMMs but also solves the label bias problem.

The main difference between CRFs and MEMMs is that a MEMM uses per-state exponential models for the conditional probabilities of next states given the current state, while a CRF has a single exponential model for the joint probability of the entire sequence of labels given the observation sequence.

Conditional Random Fields for Information Extraction

Conditional Random Fields are undirected graphical models used to calculate the conditional probability of values on designated output nodes, given values assigned to other designated input nodes.

In the special case in which the designated output nodes of the graphical model are linked by edges in a *linear chain*, CRFs make a first-order Markov independence assumption among output nodes, and thus correspond to finite state machines (FSMs). In this case, CRFs can be roughly understood as conditionally-trained hidden Markov models. CRFs of this type are a globally-normalized extension to MEMMs that avoid the label-bias problem, and are particularly suitable for sequence labeling.

Let $\mathbf{o} = \{o_1, o_2, \ldots, o_T\}$ be an observed input sequence. Let $\mathcal{S}$ be a set of FSM states, each of which is associated with a label $l \in \mathcal{L}$. Let $\mathbf{s} = \{s_1, s_2, \ldots, s_T\}$ be a state sequence. A linear-chain CRF with parameters

$\Lambda = \{\lambda, \ldots\}$ defines the conditional probability of a state sequence, given an input sequence as:

$$P_\Lambda(\mathbf{s}|\mathbf{o}) = \frac{1}{Z_\mathbf{o}} \exp\left(\sum_{t=1}^{T}\sum_{k} \lambda_k f_k(s_{t-1}, s_t, \mathbf{o}, t)\right) \tag{2.5}$$

where $Z_\mathbf{o}$ is a normalization factor over all state sequences, $f_k(s_{t-1}, s_t, \mathbf{o}, t)$ is a feature function which is often binary-valued (but can be real-valued), and λ_k is a learned weight associated with feature f_k. A feature function, in a research paper extraction task, for instance, can be defined to have value 0 in most cases, and have value 1 if and only if s_{t-1} is state #1 (which may have label TITLE), and s_t is state #2 (which may have label AUTHOR), and the observation at position t in $\mathbf{o}$ is a word appearing in a lexicon of people's first names. Large positive values for λ_k indicate a preference for such an event, while large negative values make the event unlikely.

CRFs define the conditional probability of a label sequence based on total probability over the state sequences,

$$P_\Lambda(\mathbf{l}|\mathbf{o}) = \sum_{\mathbf{s}:l(\mathbf{s})=\mathbf{l}} P_\Lambda(\mathbf{s}|\mathbf{o})$$

where $l(\mathbf{s})$ is the sequence of labels corresponding to the labels of the states in sequence $\mathbf{s}$.

Note that, the normalization factor $Z_\mathbf{o}$ (also known in statistical physics as the *partition function*) is the sum of the "scores" of all possible state sequences:

$$Z_\mathbf{o} = \sum_{\mathbf{s}\in\mathcal{S}^T} exp\left(\sum_{t=1}^{T}\sum_{k} \lambda_k f_k(s_{t-1}, s_t, \mathbf{o}, t)\right)$$

and that the number of state sequences is exponential in the input sequence length, T. In arbitrarily-structured CRFs, calculating the partition function in closed form is intractable, and approximation methods such as Gibbs sampling, or loopy belief propagation must be used. In linear-chain-structured CRFs (in use here for sequence modeling), the partition function can be calculated efficiently by dynamic programming.

The $\Lambda = \{\lambda, \ldots\}$ weights of a CRF can be estimated by maximum likelihood, maximizing the conditional probability of a set of label sequences, each given their corresponding input sequences. Maximum likelihood training chooses parameter values such that the logarithm of the likelihood, known as log-likelihood, is maximized. For a CRF, the log-likelihood L of a training set $\mathcal{D} = \{(\mathbf{o}, \mathbf{l})^{(1)}, \ldots, (\mathbf{o}, \mathbf{l})^{(N)}\}$ is given by:

$$L = \sum_{i=1}^{N} \log P_\Lambda(\mathbf{l}^{(i)}|\mathbf{o}^{(i)})$$

It is worth noticing that CRFs share many of the advantageous properties of standard maximum entropy models, including their convex likelihood function, which guarantees that the learning procedure converges to the global maximum. It is not, however, straightforward to find it quickly.

Parameter estimation in CRFs requires an iterative procedure, and some methods require fewer iterations than others. Although the original presentation of CRFs [88], described training procedures based on iterative scaling (GIS or IIS), it is significantly faster to train CRFs and other "maximum entropy"- style exponential models by a quasi-Newton method, such as *L-BFGS* [93, 128, 21] . This method approximates the second-derivative of the likelihood by keeping a running, finite window of previous first-derivatives. Sha and Pereira [128] show that training CRFs by L-BFGS is several orders of magnitude faster than iterative scaling.

To avoid overfitting, log-likelihood is often penalized by some prior distribution over the Λ parameters. Three prior distribution are used in literature: Gaussian prior [28], exponential prior [63], and hyperbolic-L_1 prior [115].

MALLET: an implementation of CRFs

CRFs are widely used in information extraction task because they offer a unique combination of properties: discriminatively trained models for sequence segmentation and labeling; combination of arbitrary, overlapping and agglomerative observation features; efficient training and decoding based on dynamic programming, and parameter estimation guaranteed to find the global optimum.

The main disadvantage of CRFs is the computational expense of training. Although CRF training is feasible for many real-world problems, the need to perform inference repeatedly during training becomes a computational burden when: there are a large number of training instances, the graphical structure is complex, there are latent variables, or the output variables have many outcomes.

Although some drawbacks, CRFs are the state-of-the-art approach in text segmentation task, and thus they represent the reference opponent for each segmentation system. In Chapter 3, a classification-based approach to text segmentation is proposed and its effectiveness is tested (not just) against a Java implementation of CRFs named *MALLET* [95].

Actually, MALLET is an integrated collection of Java code useful for statistical natural language processing, document classification, clustering, information extraction, and other machine learning applications to text. In particular, *Simple Tagger* is a command line interface to the MALLET Conditional Random Field class.

The use of Simple Tagger is really easy. The input file should be in the format as in Figure 2.12 where each line represents one token, and has the format [*feature-1, feature-2,...,feature-n, label*].

```
Bill CAPITALIZED noun
slept non-noun
here LOWERCASE STOPWORD non-noun
```

Fig. 2.12. An excerpt of input file in MALLET

Then, a CRF using Simple Tagger can be trained. This produces a trained CRF in the file "nouncrf" that can be finally used on a non-labeled test file in order to obtain the right segmentation.

2.5 Conclusions

In this chapter, an extensive (but not exhaustive) survey on techniques used for the Schema Reconciliation problem when it deals with text-related scenarios are provided. In this case, the Schema Reconciliation problem can be conducted to the wide area of *Information Extraction*.

As previously viewed, Information Extraction approaches can broadly separate in two large categories: rules-based and stochastic approaches. Moreover, the rules can be hand-coded or automatically learned from the documents.

The need of adaptive approaches originates from the endeavor to reduce the amount of hand-coded domain knowledge. However, this approaches still require human contribution in various forms depending on their class. Human knowledge is utilized as explicit knowledge sources, examples specifying what to extract by identifying relevant content in training text, or in form of human supervision interacting directly with the system during correction of the extraction proposals. While statistical and machine learning approaches rely on the latter, hand-coded approaches focus mainly on the former. Arguably, the annotation of texts is often less costly than the explicit formalization of knowledge, which makes statistical or machine learning approaches more attractive for adaptation to domains where no explicit knowledge sources are available. However, on the other hand, the hand-coded approaches allow easier re-use of existing formalizations.

Basically, rule-based approaches try to exploit the regularity in expressions of certain information to find common linguistic patterns that match these expressions. As discussed, the majority of approaches use rule learning techniques to acquire the patterns.

Statistical approaches reduce the IE task to the prediction problem. In the simple case, every text token is classified as some attribute of the target structure or not relevant. Thereby, they utilize training data very effectively being able to learn the correct prediction even from quite limited numbers of examples.

Obviously, it is difficult to determine which technique is best suited for any IE task and domain. There are many parameters that affect this decision, and the choice must be made with respect to the task to accomplish.

3

A Supervised Approach to Text Segmentation

3.1 Introduction

The extraction of "interesting information" from unstructured or loosely structured text is an actual and challenging problem. Indeed, nowadays large quantities of available data are stored in textual format. In many cases, this information has a latent schema consisting of a set of attributes, that would in principle allow to fit such textual data into some field structure, so that to exploit the mature relational technology for more effective information management. Yet in Chapter 2, an extensive review of the main techniques used in several information extraction tasks are showed.

In this chapter we propose *RecBoost*, a novel approach to that particular sub-task of the wide IE area named *Text Segmentation*. RecBoost adopts classification as an effective mechanism for segmenting free text into tuples, and further reconciling them into a common attributes structure. More in detail, RecBoost works by performing two macro-steps, namely preprocessing and reconciliation. The former step is primarily thought for formatting the individual lines of text, with potentially-different encoding format, into a uniform representation. Domain-specific ontologies and dictionaries are then exploited to associate each token with a label denoting its ontological or syntactic category. The latter step (reconciliation phase) is eventually accomplished in terms of *progressive classification*, i.e., a multi-stage classification scheme where, at each intermediate stage, a classifier is learnt from the previous classification outcome, thus being specifically targeted at handling with those textual fragments not reconciled yet.

The main contribution of the proposed approach is the introduction of a methodology realizing a strict cooperation between ontology-based generalization and rule-based classification. A key feature is the introduction of *progressive classification*, which iteratively enriches the available ontology, thus allowing to incrementally achieve accurate reconciliation. This ultimately differentiates *RecBoost* from previous works in the current literature, which adopt schemes with fixed background knowledge, and hence hardly adapt to capture the multi-faceted peculiarities of the data under investigation.

Still, the approach is further strengthened by the adoption of *local rule-based classification models*, i.e., patterns of term co-occurrence associated with specific class labels. Local models work practically well in combination with progressive classification, since they only handle the local specificities they are able to cope with, and postpone the unknown cases to subsequent classification stages. By contrast, traditional approaches from the literature exploit global classification models, which are more prone to overfitting when dealing with the several contrasting specificities occurring across individual sequences.

The outline of the chapter is as follows. Section 3.2 introduces the basic notation for the problem we face; next, it continues by covering details on the process adopted to learn a generic rule-based classifier and, then, proceeds to examine the RecBoost methodology. The architecture of RecBoost is discussed in Section 3.3. The overall RecBoost methodology is then elucidated in Section 3.4. Section 3.5 presents the results of an intensive experimental evaluation. Section 3.6 exposes a qualitative comparison with some approaches from the literature that are closely related to our study. Finally, Section 3.7 draws some conclusions and highlights a number of interesting directions, that are worth further research.

3.2 Text Segmentation with RecBoost

First of all, in order to formalize the RecBoost approach, let assume the following basic definitions. An *item domain* $\mathcal{M} = \{a_1, a_2, \ldots, a_M\}$ is a collection of *items.* Let s be a *sequence* $a_1, \ldots, a_m$ where $a_i \in \mathcal{M}$. The set of all possible sequences is denoted by $\mathcal{M}^*$. In general, an item a_k belongs to a sequence s (denoted by $a_k {\in} s$) if $s = a_1, a_2, \ldots, a_k, \ldots, a_n$. Moreover, we denote the subsequence $a_1, a_2, \ldots, a_{k-1}$ as $pre_s(a_k)$, and the subsequence $a_{k+1}, a_{k+2}, \ldots, a_n$ as $post_s(a_k)$.

A *descriptor* $R = \{A_1, \ldots, A_n\}$ is a set of attribute labels. A descriptor corresponds to a database schema, with the simplification that, for each attribute label A_i, domain information is omitted. Thus, our specific problem can be viewed as follows: given a descriptor (database relation) $R = \{A_1, \ldots, A_n\}$, and a data set of sequences (free text) $S = \{s_1, \ldots, s_m\}$, we want to segment each sequence s_i into subsequences $s_i^1, \ldots, s_i^k$, such that each token $a {\in} s_i^h$ is associated with the proper attribute A_j.

For instance, we may want to fit an unstructured collection of personal demographic information representing names, addresses, zip codes and cities, in a proper schema with specific fields for each category, as exemplified in Figure 3.1.

Text reconciliation can be profitably employed in several contexts. In Chapter 2, some of such contexts were considered: the harmonization of unformatted postal addresses collected from different data source, the processing

s_1	Harry Hacker Northern Boulevard 3001 London
s_2	C Cracker Salisbury Hill Flushing
s_3	Tony Tester Brooklyn Johnson Avenue 2

(a)

	NAME	ADDRESS	ZIP CODE	CITY
s_1	Harry Hacker	Northern Boulevard	3001	London
s_2	C Cracker	Salisbury Hill		Flushing
s_3	Tony Tester	Johnson Avenue	2	Brooklyn

(b)

Fig. 3.1. (a) Unstructured data. (b) Reconciled data.

of bibliographic records, collections of information about products, medical sheets, and so forth.

RecBoost represents an approach for contextualized reconciliation, that moves away from probabilistic modeling. The idea is to first foresee a segmentation of textual sequences into tokens and, then, to perform a token-by-token classification, that involves the analysis of the surrounding context. This basic task is at the heart of *progressive classification*, i.e., a strategy for text reconciliation, consisting in the exploitation of multiple, consecutive stages of classification. At each intermediate stage, a classifier learns from the outcome of its antecedent how to deal with those tokens, that were not reconciled at the end of the previous stage. This ensures reconciliation effectiveness even on unknown terms.

3.2.1 The RecBoost Methodology

The reconciliation of a set $S = \{s_1, \ldots, s_m\}$ of sequences with an attribute schema $R = \{A_1, \ldots, A_n\}$ consists in the association of each token a within the generic sequence $s \in S$ with an appropriate attribute of R.

RecBoost pursues text reconciliation via term generalization. Precisely, two types of generalizations are involved, namely syntactic/ontological analysis, and contextual generalization. The former aims at labelling textual tokens with their syntactic or ontological categories. The latter employs knowledge of the relationships among textual tokens, ontological categories and schema attributes, for assigning each token to a proper schema attribute.

As an example, a token a composed by multiple consecutive digits may be ontologically denoted as a number. Subsequently, the contextual presence on the same sequence containing a of two further ontological labels, such as city and street (respectively following and preceding a), may determine the reconciliation of a with an attribute address of the schema descriptor.

Syntactic and Ontological Analysis

RecBoost exploits a user-defined domain ontology, in the style of [1, 15], in order to preprocess sequences within S. In practice, a domain ontology is specified as $\mathcal{G} = \langle \mathcal{L}, \lhd, \mathcal{A} \rangle$, where $\mathcal{L}$ is a set of categories, $\lhd$ is a precedence relation defined on $\mathcal{L}$, and $\mathcal{A}$ is a set of rules whose structure is sketched below:

if *Condition*
then *Action*

Intuitively, $\mathcal{L}$ represents a set of ontological concepts, which can be exploited in order to generalize tokens within a sequence. Such concepts are structured in a concept hierarchy, specified by the $\lhd$ relation. Figure 3.2 shows an exhaustive set of concepts and their hierarchical relationships.

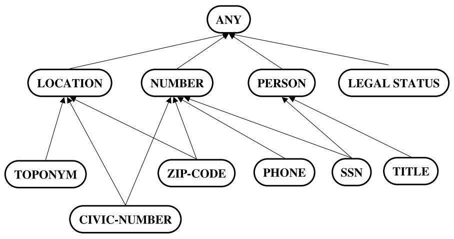


Fig. 3.2. A concept hierarchy for personal information in a banking scenario

Rules in $\mathcal{A}$ are useful to specify background knowledge about the domain under consideration, and are meant to provide a transformation of a set of tokens appearing in a sequence. It is worth noticing that, typically, the prior definition of a number of rules allows to properly deal with several tokens in a wide variety of applicative settings, thus not requiring a substantial human effort.

More specifically, *Condition* specifies a pattern-matching expression defined over the tokens of a sequence, and *Action* specifies some actions to take on such tokens. We here focus on two main actions, shown in the following illustration:

r_1: **if** a is a four-digits token
then **replace** a with ZIP-CODE

r_2: **if** a_i is a four-digits token
and a_{i+1} is a token containing digits
then **merge** a_i and a_{i+1} into a token a

Relabelling actions, such as r_1 substitute a token (or a set of tokens) with a concept in $\mathcal{L}$. *Restructuring* actions, such as r_2, operate on a set of tokens by applying basic transformation operations (such as deleting, merging or segmenting).

We also assume that some rules can exploit user-defined dictionaries. As an example, rule r_3 specifies that each token appearing in the set DICTIONARY of all known toponyms (which comprise, e.g., `street`, `road`, `blvd`, etc.) can be generalized by the category TOPONYM in $\mathcal{L}$.

$$r_3\colon \begin{array}{ll} \textbf{if} & a \in \textsc{Dictionary} \\ \textbf{then} & \textbf{replace } a \text{ with } \mathsf{TOPONYM} \end{array}$$

By exploiting $\mathcal{G}$, syntactic generalization performs two steps. First, it transforms the original sequences in $S = \{s_1, \ldots, s_n\}$ into a new set $S' = \{s'_1, \ldots, s'_n\}$, where each sequence s'_i is obtained from s_i by applying the rules in $\mathcal{A}$[1]. Second, the available tokens in each sequence are further generalized by an ad-hoc exploitation of the hierarchy described by the $\triangleleft$ relation. The exploitation is a direct result of a cooperation with contextual analysis, which reconciles tokens in S' as described in the next subsubsection.

Contextual Analysis

This step is meant to associate tokens in S with their corresponding attributes in R. We approach the problem from a supervised learning perspective. Formally, we assume that there exists a partial function $\lambda : \mathcal{M}^* \mapsto \mathcal{M} \mapsto R$ that, for each sequence $s \in \mathcal{M}^*$, labels a token a into a schema attribute A_j, namely $\lambda_s(a) = A_j \in R$. Hence, the problem can be stated as learning λ from a training set T such that, for each sequence $s \in T$ and for each token $a_i \in s$, the label $\lambda_s(a_i)$ is known.

In order to correctly classify each token $a_i \in s$, we exploit information about its *context*. The "context" of a generic token $a_i \in s$, is the set of all the items preceding and following a_i in s. Thus, we hold the context of a_i introducing the notation:

$$\mathit{feature}_s(a_i) = \langle \mathit{pre}_s(a_i),\ a_i\ , \mathit{post}_s(a_i) \rangle$$

[1] Notice that multiple matching preconditions can hold for the same set of tokens. This potential ambiguity is solved via a user-defined order over the rules in $\mathcal{A}$: when multiple rules can be applied, the first rule is chosen, and the others are ignored. In the above example, both rules r_1 and r_2 can be potentially applied to a sequence of digits. However, a token containing 4 digits can be interpreted as a zip code if and only if it is not followed by a new number (in which case, the former token has to be interpreted as an area code within a phone number). Thus, in order to disambiguate rule selection, r_2 is given a precedence on r_1, so that to initially favor the attempt at generalizing longer digit sequences.

The set $\mathcal{T} = \{\langle \mathit{feature}_s(a), \lambda_s(a)\rangle | s \in T, a \in s\}$ represents the training set for our classification problem.

The idea beyond contextual analysis is to examine the context $\mathit{feature}_s(a)$ of each token a within any sequence s, in order to learn meaningful associations among groups of tokens of S. These associations can be then exploited to learn a rule-based classifier, that associates each individual token in S with an attribute in R. In practice, our objective is to build a classifier $C : (\mathcal{M} \cup \mathcal{L} \cup R)^* \mapsto \mathcal{M} \mapsto R$, specified by rules such as the one sketched below:

$$\begin{array}{ll} \textbf{if} & \mathit{Condition} \\ \textbf{then} & \lambda_s(a) = \mathit{Class} \end{array}$$

Here, a and s represent, respectively, token and sequence variables. Moreover, *Condition* represents a conjunction of terms, and *Class* represents an attribute in R. Terms in *Condition* can be specified in three different forms: either as $a = v$, $v \in \mathit{pre}_s(a)$ or $v \in \mathit{post}_s(a)$, where v is any constant in $\mathcal{M} \cup \mathcal{L} \cup R$.

In the process of distilling a rule-based classifier from a training set T, a holdout approach is adopted to partition T into a validation set V and an actual training set $D = T - V$. The goal is learning a classifier from D that has highest accuracy on V. In principle, any rule-based classifier could be used here. However, we found that classification based on association rules is more effective in this setting than, e.g., traditional algorithms based on decision-tree learning. The intuition behind the above statement is that association rules are better suited to detect local patterns which hold "locally" on small subsets of D. This is especially true when D is large, and contains many contrasting specificities across individual sequences. By contrast, decision trees represent global models, which are hardly able to capture such specificities without incurring into the overfitting phenomenon. In addition, the intrinsic unstructured nature of the feature space to analyze does not allow an immediate application of decision-tree learning techniques, whereas association rule mining techniques naturally fit the domains under consideration.

A variant of the Apriori algorithm [136] is exploited to extract from the explicit representation of token contexts, $\mathcal{D} = \{\langle \mathit{feature}_s(a), A\rangle | s \in D, a \in s, A \in R\}$, a set of association rules that meet pre-specified requirements on their support and confidence values and whose consequents are narrowed to individual schema attributes. A classifier can hence be built on the basis of such discovered rules, by selecting the most promising subset, i.e, the subset of rules which guarantees the maximal accuracy. To this purpose, we adopted the CBA-CB method [92], which allows an effective heuristic search for the most accurate association rules. Succinctly, its basic idea is to sort the extracted associations by exploiting a precedence operator $\prec$. Given any two rules r_i and r_j, r_i is said to have a higher precedence than r_j, which is denoted by $r_i \prec r_j$, if (i) the confidence of r_i is greater than that of r_j, or (ii) their confidences are the same, but the support of r_i is greater than that of r_j, or (*iii*) both confidences and supports are the same, but r_i is shorter than r_j. Hence, a classifier can be formed by choosing a set of high precedence rules such that:

1. each case in the training set $\mathcal{D}$ is covered by the rule with the highest precedence among those that can actually cover the case.
2. every rule in the classifier correctly classifies at least one case in $\mathcal{D}$, when it is chosen.

The resulting classifier can be modelled as $\langle r_1, r_2, \ldots, r_n \rangle$, where $r_i \in \mathcal{D}$, $r_a \prec r_b$ if $b > a$. While considering an unseen case of $\mathcal{D}$, the first rule that covers the case also classifies it. Clearly, if no rule applies to a given case, the case is unclassified.

We revised the scheme of [92] by implementing a post-processing strategy, which aims at (1) further improving the classification accuracy of the discovered rules, and at (2) reducing the complexity of the discovered rules. The postprocessing is mainly composed by attribute and rule pruning. The idea behind attribute pruning consists in removing items from classification rules, whenever this does not worsen the error rate of the resulting classifier. The validation set V is exploited to assess classification accuracy.

Precisely, let r be a generic classification rule containing at least two terms in the antecedent. Also, assume that s denotes a generic sequence in V and that x represents a token within s. The error r_x of rule r on x is a random variable:

$$r_x = \begin{cases} 1 \text{ if } r \text{ misclassifies } x \\ 0 \text{ otherwise} \end{cases}$$

Hence, the overall error of r on V can be defined as follows:

$$E(r) = \frac{1}{n_V} \sum_{x,s/x \in s, s \in V} r_x$$

where n_V indicates the overall number of tokens within V. A new rule r' can now be generated by removing from the antecedent of r any of its terms. We replace r by r' if two conditions hold, namely $E(r') < E(r)$ and the discrepancy $E(r) - E(r)'$ is statistically relevant. To verify this latter condition, we exploit the fact that for n_V large, the distribution of $E(r)$ approaches the normal distribution. Hence, we compute a $\tau\%$ confidence interval $[\alpha, \beta]$, whose lower and upper bounds are respectively given by:

$$\alpha = E(r) - c_\tau \sqrt{\frac{E(r)[1 - E(r)]}{n_V}}$$

and

$$\beta = E(r) + c_\tau \sqrt{\frac{E(r)[1 - E(r)]}{n_V}}$$

where, constant c_τ depends on the confidence threshold τ. The above interval represents an estimate for the actual error of rule r. Finally, we retain r' instead of r, if it holds that $E(r') < \alpha$. In such a case, we analogously proceed to attempt at pruning further items from the antecedent of r'. Otherwise, we reject r'.

Rule pruning instead aims at reducing the number of rules in a classifier. As in the case of attribute pruning, the idea consists in removing rules from a classifier, whenever this does not worsen the accuracy of the resulting classifier.

To this purpose, all rules in a classifier are individually evaluated on the basis of their precedence order. A generic rule r is removed, if one of the following conditions holds:

- r does not cover a minimum number of cases in V.
- the accuracy of r on V is below a minimum threshold.
- the removal of r from the classifier increases its overall accuracy on V.

3.3 RecBoost Anatomy

Association rules for classification allow to tune the underlying classification model to a local sensitivity. However, in principle their adoption can yield a high number of *unclassified tokens*, i.e., tokens for which no rule precondition holds. In a reconciliation scenario, this is due to the presence of unknown or rare tokens, as well as errors in the text to segment. The adoption of a concept hierarchy mitigates such a drawback and, indeed, it has already been adopted in traditional approaches based on HMM [1, 15]. The novelty in the RecBoost reconciliation methodology relies on a finer cooperation between synthactic/ontological analysis and contextual analysis. The reiteration of the process of transforming tokens and learning a rule-based classifier allows *progressive classification*, i.e., the adoption of multiple stages of classification for more effective text reconciliation. Precisely, a pipeline $\mathcal{C} = \{C_1, \ldots, C_k\}$ of rule-based classifiers is exploited to this purpose. At the generic step i, $i = 2, \ldots, k$, a classifier C_i is specifically learnt to classify all those tokens, that were not reconciled at the end of step $i - 1$. The length k of the classification pipeline is chosen so that to achieve accurate and exhaustive classification. Conceptually, this requires to minimize the overall number of both misclassified and unclassified tokens. In practice, a further classification stage is added to P whenever such values do not meet application-specific requirements, such as in the case where the misclassification rate is acceptable, but the unclassification rate is not satisfactory.

The generic classifier C_i can be formally described as a partial mapping $C_i : (\mathcal{M} \cup \mathcal{L} \cup R)^* \mapsto \mathcal{M} \mapsto R$, and its construction relies on a specific training set T_i, that is obtained from T_{i-1} by adding domain information provided by

C_{i-1}. Given any sequence $s \in T_{i-1}$, C_i is learnt from the evaluation of the set X_s of unknown tokens, i.e., the set of those tokens in s, that are not covered by any rule of C_{i-1}. This is accomplished by enriching the domain information in $\mathcal{G}$ with a new set of rules directly extracted from the set of classification rules in C_{i-1}. Specifically, each classification rule $r \in C_{i-1}$ such as the one below:

if *Condition*
then $\lambda_s(a) = \mathit{Class}$

is transformed into a labelling rule r', having the following structure:

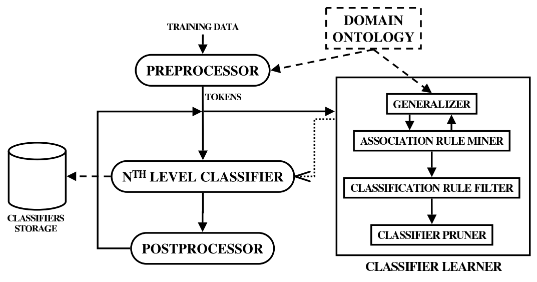
if *Condition*
then **replace** a with *Class*

The new rule r' is then added to the set $\mathcal{A}$ of rules available for syntactic analysis. Then, syntactic analysis is applied to each sequence s in T_{i-1}, and the resulting transformed sequences are collected in T_i. A new training set $\mathcal{T}_i$ is then generated by collecting, for each sequence $s \in T_i$ and each token $a \in X_s$, the tuples $\langle \mathit{feature}_s(a), \lambda_s(a) \rangle$. Notice that there is a direct correspondence between the context $\mathit{feature}_s(a)$ computed at step i and the context computed at step $i-1$. Indeed, the new context $\mathit{feature}_s(a)$ follows from the context of a within T_{i-1} by replacing each token $b \notin X_s$ of s with its corresponding attribute $C_{i-1}(b)$.

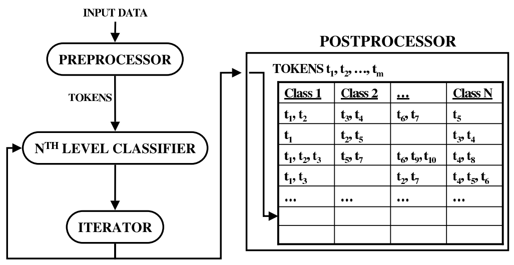
The above detailed methodology is supported by three main components, namely a *preprocessor* (*tokenizer*), a *classifier learner* and a *postprocessor.* The components cooperate both in the training and in the classification phases, as detailed in Figure 3.3. In the following, we explain the role played by each of the aforementioned modules.

3.3.1 Preprocessor

A cleaning step is initially performed by this component, to the purpose of encoding the initial data sequences of a free text S into a uniform representation. This phase involves typical text-processing operations, such as the removal of stop-words, extra blank spaces, superfluous hyphens and so forth. The *preprocessor* then proceeds to split free text into tokens. The main goal of this phase is to recognize domain-dependent symbol-aggregates (e.g. acronyms, telephone numbers, phrasal construction, and so on) as single tokens. As an example, aggregates such as '*I B M*', '*G. m. b. H.*' or '*as well as*' are more significant as a whole, rather than as sequences of characters in the text. The identification of symbol aggregates as well as domain/specific cleaning steps are accomplished by using domain-specific transformation rules suitably defined in $\mathcal{G}$.



(a)



TOKENS $t_1, t_2, \ldots, t_m$

Class 1	Class 2	...	Class N
t_1, t_2	t_3, t_4	t_6, t_7	t_5
t_1	t_2, t_5		t_3, t_4
t_1, t_2, t_3	t_5, t_7	t_6, t_9, t_{10}	t_4, t_8
t_1, t_3		t_2, t_7	t_4, t_5, t_6
...	...	...	...

(b)

Fig. 3.3. Training (a) and Classification (b) phases in the RecBoost methodology.

3.3.2 Classifier Learner

The *classifier learner* is responsible for producing an optimal set of classification rules, as shown in Figure 3.3(a). It consists of four main elements: a *generalizer*, an *association rule miner*, a filter for *classification rules* and a *classifier pruner*. In particular, the *generalizer* performs ontological generalization, by exploiting the labelling rules and the $\lhd$ relationship defined in $\mathcal{G}$. Its role is mainly to enable the discovery of accurate association/classification rules, by providing an adequate degree of generalization among the data. To accomplish this task, the generalizer employs the labelling rules in $\mathcal{A}$. Next, for each label replacing a token somewhere in a textual sequence, the related concept hierarchy is inspected and the textual sequence is extended to also include the ancestors of the specific label. The latter operation is performed by the *association rule miner*, that extracts generalized association rules from the above extended sequences. The classification rules filtered by the *classification rules filter*, which in principle could contain several redundancies (due to the exploitation of the hierarchy in the association mining step), are further postprocessed by the *classifier pruner*. The latter attempts to reduce the overall size of the discovered rules by exploiting the aforementioned attribute and rule pruning techniques.

3.3.3 Postprocessor

The *postprocessor* rebuilds the sequences reconciled by a rule-based classifier, at any stage of progressive classification, by fitting them into a relational structure with schema R, as shown in Figure 3.3(b). This is accomplished by interpreting each (partially) reconciled sequence as a structured tuple, and organizing the tokens that have been so far reconciled as values of corresponding schema attributes.

Postprocessing enables progressive reconciliation: at any stage, a classifier is specifically learnt for dealing with those sequence tokens, that were not reconciliated at the end of the previous stage. The postprocessor is also exploited during the training phase, as shown in Figure 3.3, to yield the i-th training set T_i, by generalizing the tokens in each sequence $s \in T_{i-1}$ via the application of the rules in C_{i-1}.

3.4 An Illustrative Example

Here, we elucidate the overall RecBoost methodology, by exemplifying the reconciliation of a collection of personal demographic information, shown below, in compliance with the attribute descriptor $R = \{\mathsf{NAME}, \mathsf{ADDRESS}, \mathsf{ZIP}, \mathsf{CITY}\}$.

s_1	Harry Hacker 348.2598781 "Northern - Boulevard" (3001) London
s_2	C. Cracker ... Salisbury Hill, Flushing
s_3	Tony Tester Johnson Avenue 2 -Brooklyn- 323-45-4532

In particular, we assume to exploit a dictionary D, containing all known toponyms, and a domain-specific ontology $\mathcal{G} = \langle \mathcal{L}, \triangleleft, \mathcal{A} \rangle$, such that $\mathcal{A}$ consists of the following ontological rules:

r_1:	**if**	a is a four-digits token
	then	**replace** a with ZIP-CODE
r_2:	**if**	a is a token of more that four digits
	then	**replace** a with PHONE-NUMBER
r_3:	**if**	a is a token of type $ddd - dd - dddd$,
	and	d is a digit
	then	**replace** a with SSN
r_4:	**if**	$a \in$ DICTIONARY
	then	**replace** a with TOPONYM

The example data collection is corrupted by noise, i.e. by the absence of a uniform representation for all of its constituting sequences. Indeed, a comparative analysis of their formatting encodings reveals that:

- there is a telephone number in sequence s_1 that has to be discarded, since it is not expected by the descriptor R.
- character '-' is employed in sequence s_1 as a separator between the words Northern and Boulevard, that are in turn delimited by double quotes.
- brackets are exploited to separate the zip-code information in sequence s_1.
- three non-relevant dots precede the address information in sequence s_2.
- two hyphens in sequence s_3 demarcate the word Brooklyn.
- there is a social security number (SSN) in sequence s_3 that has to be discarded, since it is not expected by the descriptor R.

The identification of a uniform representation format for all of the individual sequences in the textual database enables an effective segmentation of such sequences into tokens and, hence, a reliable reconciliation. A preprocessing step is performed to this purpose.

3.4.1 Preprocessing

The input textual sequences are suitably tokenized. This is accomplished by exploiting the presence in the original text of domain-specific delimiters such as single or double quotes, hyphens, dots, brackets and blanks. After segmentation, such delimiters become spurious characters, i.e. play no further role in the reconciliation process, and are hence ignored.

The output of this step, with respect to the hypothesized data, is represented below:

s_1	Harry	Hacker	3482598781	Northern	Boulevard	3001	London
s_2	C	Cracker	Salisbury	Hill	Flushing		
s_3	Tony	Tester	Johnson	Avenue	2	Brooklyn	323-45-4532

The fragmented text is now subjected to a pipeline of rule-based classifiers, that reconciliate groups of tokens across the individual sequences s_1, s_2, s_3 with the attributes in R.

For the sake of convenience, we assume that two stages of classification allow the accomplishment of an actual reconciliation. Furthermore, since progressive classification involves a similar processing for each sequence in the tokenized text, we proceed to exemplify the sole reconciliation of s_1.

3.4.2 Progressive Classification

Progressive classification divides into syntactic and contextual analysis.

Syntactic Analysis

This step performs token generalization. Here, the exploitation of the above ontological rules allow the generalization of a number of tokens in s_1 as shown below, where labels denoting ontological categories are enclosed between stars.

Harry	Hacker	*PHONE*	*TOPONYM*	Boulevard	*ZIP*	London

To this point, s_1 undergoes two levels of contextual analysis, where at each level, a suitable set of rules is applied.

First-level Classifier

A classifier is generally distilled from the analysis of the relationships among textual tokens, ontological categories and, also, attributes in the context of each token within the generalized sequences at hand. In particular, we suppose that the classifier resulting from the learning phase includes the classification rules listed below:

$$r_5\colon \begin{array}{ll} \textbf{if} & pre_s(a) = \emptyset \quad \wedge \\ & \{*TOPONYM*, *ZIP*\} \in post_s(a) \\ \textbf{then} & \lambda(a) = NAME \end{array}$$

$$r_6\colon \begin{array}{ll} \textbf{if} & a = *TOPONYM* \\ \textbf{then} & \lambda(a) = ADDRESS \end{array}$$

$$r_7\colon \begin{array}{ll} \textbf{if} & \{*TOPONYM*\} \in pre_s(a) \quad \wedge \\ & \{*ZIP*\} \in post_s(a) \\ \textbf{then} & \lambda(a) = ADDRESS \end{array}$$

$$r_8\colon \begin{array}{ll} \textbf{if} & a = *ZIP* \\ \textbf{then} & \lambda(a) = ZIP \end{array}$$

$$r_9\colon \begin{array}{ll} \textbf{if} & \{*TOPONYM*, *ZIP*\} \in pre_s(a) \quad \wedge \\ & post_s(a) = \emptyset \\ \textbf{then} & \lambda(a) = CITY \end{array}$$

The first-level classification hence starts by classifying the tokens of s_1, according to their features. In particular, being s_1 composed of six tokens, a first-level classifier is applied against the six context representations $feature_{s_1}(a) = \langle pre_{s_1}(a),\ a\ , post_{s_1}(a)\rangle$, shown below, where a is any token of s_1.

PRE	WORD	POST
-	Harry	Hacker TOPONYM Boulevard ZIP London
Harry	Hacker	TOPONYM Boulevard ZIP London
Harry Hacker	TOPONYM	Boulevard ZIP London
Harry Hacker TOPONYM	Boulevard	ZIP London
Harry Hacker TOPONYM Boulevard	ZIP	London
Harry Hacker TOPONYM Boulevard ZIP	London	-

Notice that, at this stage of contextual analysis, s_1 does not include attribute labels. Hence, reconciliation takes into account relationships among ontological labels and textual tokens. These enable the reconciliation of the entities *TOPONYM*, Boulevard, London and *ZIP*, but fail in dealing with *PHONE* and Hacker. In particular, this latter token is not covered by the rule that classified Harry, since $pre_{s_1}(\text{Hacker}) = \{\text{Harry}\} \neq \emptyset$. At the end of this step of classification, sequence s_1 assumes the following form:

[NAME]	Hacker	*PHONE*	[ADDRESS]	[ADDRESS]	[ZIP]	[CITY]

where reconciliated tokens are replaced by their corresponding attribute labels, enclosed between square brackets.

Second-level classifier

Contextual analysis is reiterated to reconciliate those tokens that were not associated with a schema attribute at the end of the previous step. Again, we assume that a second-level classifier is learnt from the training data, and it is composed by the sole rule:

$$r_{10}: \begin{array}{ll} \textbf{if} & \{NAME\} \in pre_s(a) \quad \wedge \\ & \{ADDRESS, ZIP\} \in post_s(a) \\ \textbf{then} & \lambda(a) = NAME \end{array}$$

There are only two tokens in s_1 that were not associated with a schema attribute and, hence, the above classifier is applied against two context representations:

PRE	WORD	POST
[NAME]	Hacker	*PHONE* [ADDRESS] [ADDRESS] [ZIP] [CITY]
[NAME] Hacker	*PHONE*	[ADDRESS] [ADDRESS] [ZIP] [CITY]

As a result, the classifier further generalizes s_1 into the following sequence:

[NAME]	[NAME]	*PHONE*	[ADDRESS]	[ADDRESS]	[ZIP]	[CITY]

Notice that *PHONE* is still not reconciliated, since no classification rule applies to it.

3.4.3 Postprocessor

The postprocessor rebuilds the original sequence s_1, by fitting its corresponding tokens in a suitable structure defined by the descriptor $R = \{\mathsf{NAME}, \mathsf{ADDRESS}, \mathsf{ZIP}, \mathsf{CITY}\}$:

NAME	ADDRESS	ZIP	CITY
Harry Hacker	Northern Boulevard	3001	LONDON

Notice that the structure above, exactly complies with R. However, in some cases, it may be useful to add an extra column NOISE, to the purpose of tracing all the original tokens. This would correspond to the following tuple:

NAME	ADDRESS	ZIP	CITY	(NOISE)
Harry Hacker	Northern Boulevard	3001	London	3482598781

3.5 Experimental Evaluation

In this section, we describe the experimental evaluation we performed on the proposed methodology. Experiments were mainly aimed at evaluating the effectiveness of the proposed methodology in segmenting strings. To this purpose, we accomplish the following tasks:

1. We evaluate the effectiveness of the basic rule-based classifier systems proposed in Section 3.2.1. Since the classification methodology represents the basic infrastructure upon which the RecBoost system bases, it is important to assess its effectiveness in the domain at hand. In particular, we evaluate two main aspects: (*i*) its dependency from the parameters which are needed to tune the system, and (*ii*) the effectiveness of the pruning strategy introduced.
2. Next, we evaluate classification accuracy obtained by the progressive classification methodology nested in the RecBoost approach, as described in Section 3.3. Our aim here is to investigate in which respect the envisaged pipeline boosts the performance of a basic classifier. We also compare our results with other state-of-the art text segmentation systems.

3.5.1 Experimental setup

In order to accomplish the above tasks, we considered the following datasets:

- `Addresses`, a real-life demographic database consisting of information about the issue-holders of credit situations in a banking scenario. Such a dataset is of particular interest, since it contains several fragments of noisy data. The dataset is of 24,000 sequences, with an average of 8 tokens per sequence. The schema to reconcile consists of the fields: *Name*, *Address*, *Zip*, *State/Province*, and *City*.

- `BigBook`, a publicy-available dataset[2] consisting of a set of business addresses. Each business description consists of the 6 items *Name*, *Address*, *City*, *State*, *AreaCode*, and *Phone*. The dataset consists of 4,224 sequences, with 10 tokens per sequence in the average. The dataset is of particular interest, since the relatively small size of the available dataset allows us to evaluate whether RecBoost is sensitive to the number of training tuples.
- `dblp`, a collection of articles extracted from the DBLP website[3]. Each entry refers to an article appeared in a Computer Science Journal, and contains information about *author*, *title*, *journal*, *volume*, *year*. We extracted 19,401 sequences, with an average sequence length of 20 tokens.

The evaluation of Recboost effectiveness requires the design of a domain-specific ontology for each of the aforementioned datasets. Specifically, the concept hierarchy devised for the `Addresses` dataset is shown in Figure 3.2. This consists of 11 concepts for token generalization, suitably organized into a compact hierarchical structure. The ontological rules include rules r_1, r_2, r_3 and r_4 at Section 3.4 (as relabelling rules) and rule r_2 at Section 3.2.1 (as a restructuring rule).

The ontology employed for the `BigBook` dataset, shown in Figure 3.4, embraces 9 concepts.

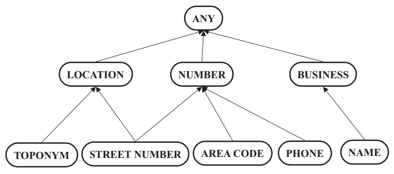


Fig. 3.4. The concept hierarchy for the `BigBook` dataset

In such a context, no use is made of restructuring actions, so that background knowledge reduces to the relabelling rules shown next:

[2] http://www.isi.edu/info-agents/RISE/repository.html
[3] http://www.informatik.uni-trier.de/~ley/db/

r_1: **if** $a \in$ DICTIONARY
then **replace** a with TOPONYM

r_2: **if** a is a token of type *ddd-dddd*
then **replace** a with PHONE NUMBER

r_3: **if** a is a token of type *(ddd)*
then **replace** a with AREA-CODE

r_4: **if** a is a digit-sequence followed by *TH* or *ST* or *ND* or *RD*
then **replace** a with STREET NUMBER

r_5: **if** a is a digit-sequence
then **replace** a with NUMBER

Finally, the ontology for the `dblp` dataset is shown in Figure 3.5.

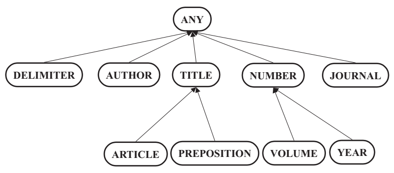


Fig. 3.5. The concept hierarchy for the `dblp` dataset

Again, background knowledge only involves relabelling rules, that are reported below:

r_1: **if** $a \in$ JOURNAL DICTIONARY
then **replace** a with JOURNAL

r_2: **if** a is a four-digits token
and $a \in [1950, 2006]$
then **replace** a with YEAR

r_3: **if** a is a digit-sequence
then **replace** a with NUMBER

r_4: **if** $a \in$ DELIMITER DICTIONARY
then **replace** a with DELIMITER

r_5: **if** $a \in$ GRAMMATICAL-ARTICLE DICTIONARY
then **replace** a with ARTICLE

r_6: **if** $a \in$ PREPOSITION DICTIONARY
then **replace** a with PREPOSITION

Notice that the definition of the above rules relies on a number of domain-specific dictionaries. In particular, JOURNAL DICTIONARY includes several alternative ways of denoting a journal article, such as *j.*, *journal*, *trans.* and *transaction.* GRAMMATICAL-ARTICLE DICTIONARY groups English-language articles *a*, *an* and *the.* Similarly, PREPOSITION DICTIONARY collects commonly used prepositions, such as *by*, *to*, *with* and *via.* DELIMITER DICTIONARY is a set of token delimiters, that comprises *'*, *"*, *;*, *,*, *-*, *.*, and ***.

It is worth noticing that the analysis of the above domain-specific ontologies reveals a key feature of RecBoost methodology. Roughly speaking, it can be easily employed for pursuing text reconciliation in a wide variety of applicative settings, by simply providing a domain-specific concept hierarchy along with a corresponding compact set of ontological rules. The overall process of ontology design is rather intuitive and does not require substantial effort by the end user.

The evaluation of the results relies on the following standard measures which are customized to our scenario. Given a set N of tokens to classify, we define:

- the number of tokens, which were classified correctly, TP.
- the number of tokens, which were misclassified, FP.
- the number of tokens, which were not classified, FN - notice that this is a different meaning with respect to the standard literature.

In the following, we shall report and illustrate the above measures over the mentioned datasets. Two further important measures, however, can give an immediate and summarizing perception of the capabilities of our classification system. In particular, *Precision* (or *Accuracy*) can be defined as the number of correctly classified tokens, w.r.t. the classification behavior of the system:

$$P = \frac{TP}{TP + FP}$$

Analogously, *Recall* can be defined as the number of correctly classified tokens, w.r.t. the tokens to classify:

$$R = \frac{TP}{TP + FN}$$

Intuitively, Recall describes the locality issues which affect the system: if a classifier contains rules which can cover all the examples, then it has 100% recall (i.e., no locality effect). Precision, by the converse, describes the accuracy of the rules contained: the higher is the error rate of a rule, the lower is its precision.

A measure which summarizes both precision and recall is the $\mathcal{F}$ measure, defined as:

$$\mathcal{F} = \frac{(\beta^2 + 1)PR}{P + \beta^2 R}$$

The $\mathcal{F}$ represents the harmonic mean between Precision and Recall. The β term in the formula assigns different weights to the components: when $\beta = 1$

both the components have the same importance. The tuning of the β parameter is application-dependent. Here, we are interested in the cases where $\beta > 1$ (which assigns higher importance to Precision than to Recall). This is a crucial requirement of many application domains where text segmentation applies. Tuple disambiguation can be accomplished either by exploiting exact matching techniques based on specific segments of the strings, or fuzzy techniques based on the entire string. Clearly, exact matching is more reliable, provided that the original text is correctly segmented. Consider, e.g., the strings:

s_1	Jeff, Lynch, Maverick, Road, 181, Woodstock
s_2	Jeff, Alf., Lynch, Maverick, Rd, Woodstock, NY

which clearly represent the same entity. A correct segmentation of the strings would eventually ease the task of recognizing the similarity:

	NAME	ADDRESS	CITY	STATE
s_1	Jeff, Lynch	`Maverick, Road, 181`	`Woodstock`	
s_2	Jeff, Alf., Lynch	`Maverick, Rd`	`Woodstock`	`NY`

However, a wrong segmentation would make things rather complicated,

	NAME	ADDRESS	CITY	STATE
s_1	Jeff, Lynch Maverick	Road	181	Woodstock
s_2	Jeff, Alf.	Lynch Maverick, Rd	Woodstock	NY

whereas simpler fuzzy techniques, which do not consider segmentation, could still resolve the ambiguity in an acceptable way.

It is clear that classification systems exhibiting high precision, even at the cost of low recall, can be safely embedded into the application scenarios described so far. Hence, in the following we shall study the situations where $\beta > 1$, and in particular we are interested in the cases where β ranges into the interval $(1, 10]$.

3.5.2 Evaluating The Basic Classifier System

In an initial set of experiments, we classified the data without exploiting ontologies and multiple classification stages. In these trials, support was fixed to 0.5%, with ranging values of confidence. Figure 3.6 shows the outcome of classification for the three datasets. Each bar in the graph describes the percentage of correctly classified tokens, together with the percentages of misclassified and unclassified tokens. As we can see, the effectiveness of the classifiers strongly relies on the confidence value. In particular, low confidence values (up to 40% in both `Addresses` and `dblp`, and 60% in `BigBook`) allow to classify all the tokens, but the percentage of misclassified is considerably high. This is somehow expected, since low confidence values induce rules exhibiting a weak correlation between the antecedent and the consequent.

By contrast, higher confidence levels lower the misclassification rate, but the degree of unclassified tokens raises considerably. It is worth noticing that, in all the examined cases a confidence rate of 100% guarantees a percentage of misclassified data which is nearly zero. This is the *locality effect*: high confidence values produce extremely accurate rules that, as a side effect, apply only to a limited number of tokens. By lowering the confidence, we relax the locality effect (the resulting rules apply to a larger number of tokens), but the resulting rules are less accurate.

The `dblp` dataset is particularly interesting to investigate in this context, since it exhibits the worst performances. The best we can obtain in this dataset is with confidence set to 40%, which guarantees a significantly high percentage (30.52%) of misclassified tokens. A "safer" confidence value leverages the number of unclassified tokens considerably.

Figure 3.7 describes the accuracy of the classifier with the adoption of domain-specific concept hierarchies. We exploited the hierarchies described in Figures 3.2, 3.4 and 3.5 respectively. The benefits connected with the exploitation of such simple ontologies are evident: the generalization capabilities of the classification rules are higher, thus lowering the number of unclassified tokens. Notice how the `dblp` dataset still exhibits unacceptable performances.

Results in Figure 3.7 were obtained also by exploiting the pruning steps detailed in Section 3.2.1. Indeed, the contribution of the classifier pruner to the misclassification rate is investigated in Table 3.1, which describes how the error rate changes if pruning is not applied. The effectiveness of the classifier pruner can be appreciated at lower confidence values: there, the classifier produces weaker rules, which clearly benefit of a re-examination.

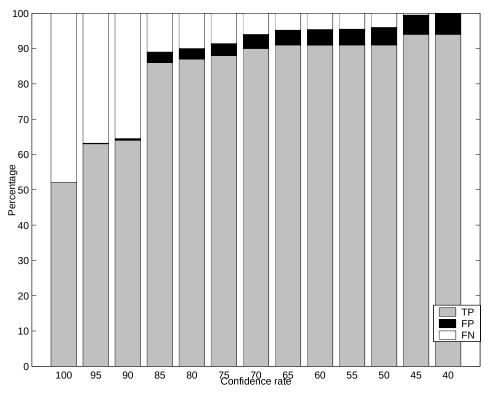
Confidence			**100**	**90**	**80**	**70**	**60**	**50**	**40**
FP	`Addresses`	*Unpruned*	0.11%	1.73%	3.93%	5.52%	6.45%	7.69%	8.05%
		Pruned	0.09%	1.53%	3.77%	5.32%	5.94%	6.47%	6.24%
	`BigBook`	*Unpruned*	0.25%	1.47%	1.50%	1.53%	1.94%	1.94%	1.95%
		Pruned	0.21%	0.70%	0.70%	0.72%	0.98%	0.98%	0.98%
	`Dblp`	*Unpruned*	0.01%	3.30%	3.81%	7.13%	14.55%	17.02%	17.12%
		Pruned	0.01%	3.26%	3.77%	7.09%	14.38%	16.06%	16.20%

Table 3.1. Pruning effectiveness

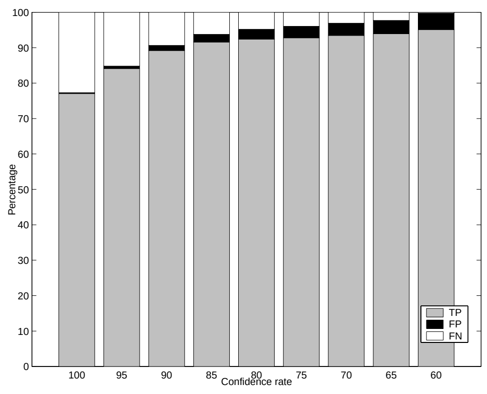
3.5.3 Evaluating Multiple Classification Stages

The above analysis allows us to test the effectiveness of the *progressive classification* methodology. We recall the underlying philosophy: starting from the following observations,

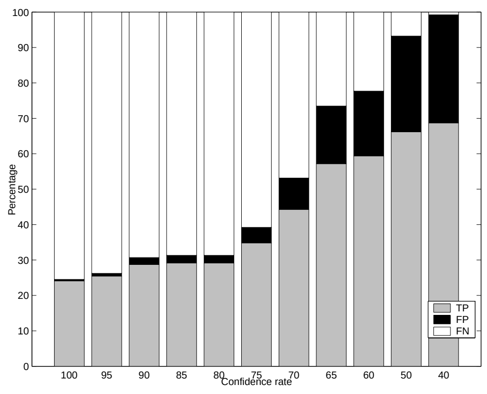
- ontological analysis eases the classification task (as testified by the comparison between graphs in Figures 3.6 and 3.7).



(a) `Addresses`

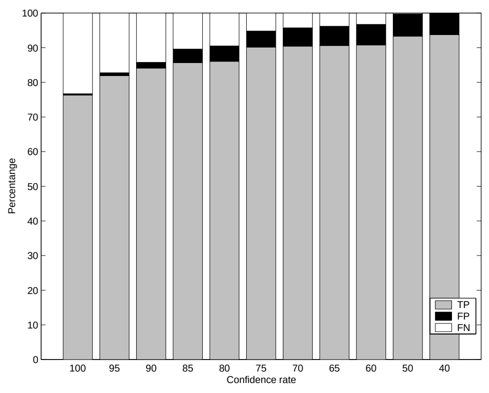


(b) `BigBook`

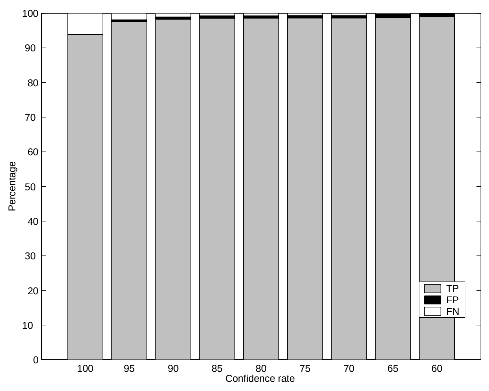


(c) `dblp`

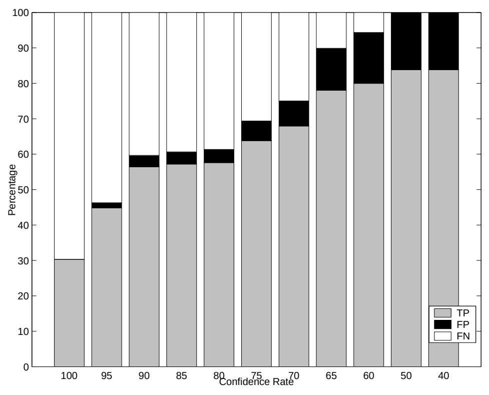
Fig. 3.6. Classification results, single stage of classification



(a) `Addresses`



(b) `BigBook`



(c) `dblp`

Fig. 3.7. Classification results with the exploitation of concept hierarchy

- a richer set of relabeling rules should in principle boost the results of classification.

the adoption of multiple classification stages, where at each stage the relabeling rules of the previous stage are enriched by exploiting the results of classification at earlier stages, should boost the performance of the overall classification process.

And indeed, Figure 3.8 describes the results obtained by applying a second-level classifier to the unclassified cases of the first stage of classification. In detail, the input to the second-level classifier is the output of the first-level classifier, built by fixing support to 0.5% and a confidence to 100% (described by the first bar of each graph in Figure 3.7). Again, support was set to 0.5% and confidence was ranged between 100% and 80%.

As shown in this figure, the second-level classifier is in general able to correctly classify a portion of the data, that were unlabelled at the end of the previous stage. For example, in the `Addresses` dataset, a 95% threshold allows to classify a further 62% of the (originally unclassified) data. By combining such a result with the outcome of the first-level classifier, we obtain nearly 91% of correctly classified data, less than 1% of misclassified data and nearly 8% of unclassified data. Table 3.2 summarizes the the cumulative results achieved by two levels of classification over the employed datasets.

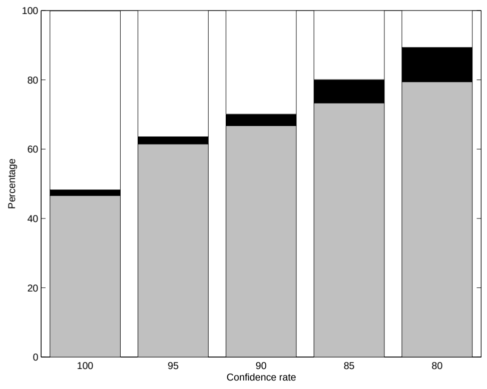
Confidence	`Addresses`		`BigBook`		`dblp`	
	P	**R**	**P**	**R**	**P**	**R**
100	99,13%	87,87%	99,78%	94,95%	99,73%	41,40%
95	99,06%	91,44%	99,68%	96,30%	97,51%	55,52%
90	98,74%	92,96%	99,57%	97,13%	94,59%	66,75%
85	97,95%	95,26%	99,56%	97,17%	93,92%	69,72%
80	97,24%	97,45%	99,39%	97,93%	91,89%	76,80%

Table 3.2. Precision and recall at varying degrees of confidence over the selected datasets

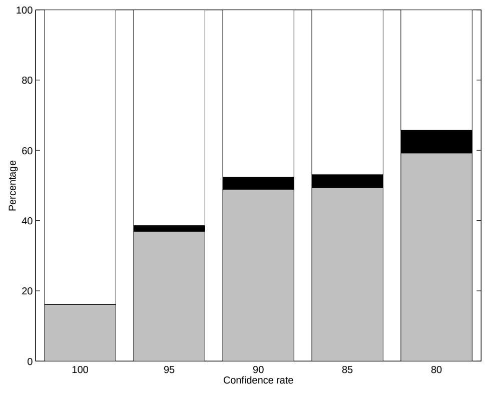
The effectiveness of the second stage of classification is even more evident in the graphs of Figure 3.9. The graphs depict the trend of $\mathcal{F}$ for different values of β. The graphs compare a selection of 2-level classifiers with the single-level classifier (among those shown in Figure 3.7) exhibiting the best performance in terms of TP. In all the cases shown, the 2-level classifiers exhibit better performances for $\beta > 2$.

Since each classification level boosts the performance of the system, two important questions raise, that are worth further investigation in the following:

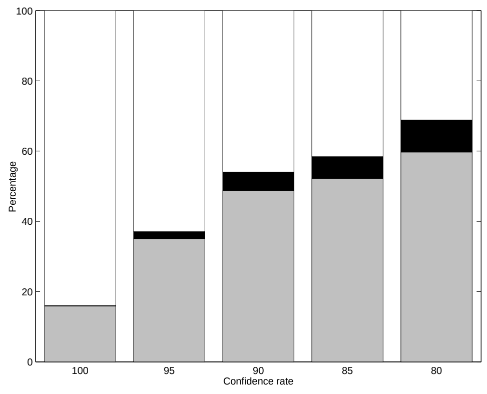
1. how many levels allow to achieve an adequate performance?
2. how should the parameters at each level be tuned?



(a) `Addresses`

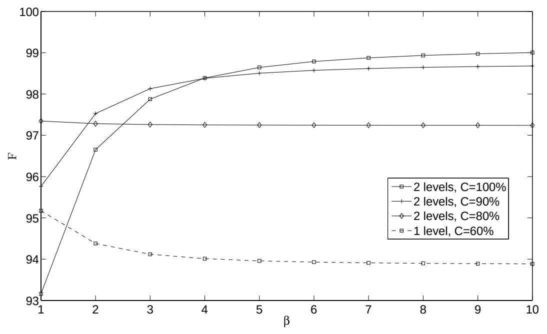


(b) `BigBook`

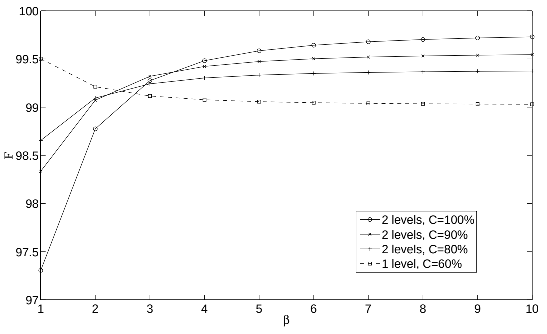


(c) `dblp`

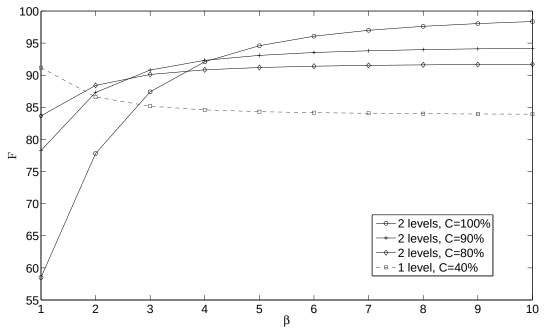
Fig. 3.8. Classification results at the second stage of classification



(a) `Addresses`



(b) `BigBook`



(c) `dblp`

Fig. 3.9. Trends of $\mathcal{F}$-measures compared

The `dblp` dataset is particularly interesting in this context, since the accuracy of RecBoost is still low after two classification levels. We start our study by investigating the number of needed classifiers. Figures 3.10(a) and (b) describe an experiment performed by allowing a hypothetical infinite number of levels, where at each level, support was set to 1% and confidence to 100%. Roughly, the strategy implemented is the following: since high confidence values bound the number of misclassified tokens, and further levels allow to recover unclassified tokens, just allow any number of levels, until the number of unclassified tokens is nearly 0.

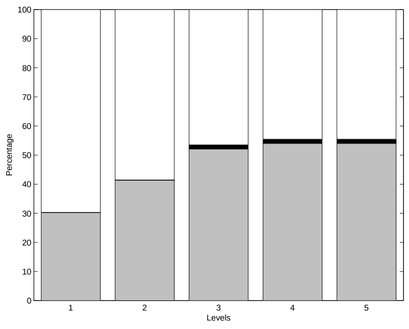
As we can see from Figure 3.10(b), however, this strategy does not necessarily work: although the number of misclassified tokens is kept low, the capability of each classifier to recover tokens unclassified in the previous stages decreases. The 5th level looses the capability to further classify tokens, thus ending de-facto the classification procedure. Figure 3.10(a) shows the cumulative results at each level.

Thus, an upper bound, in the number of stages, can be set by the classification capability of the stages themselves. A smarter tuning of the parameters which rule the performance of each single stage, allows to achieve best classification accuracy. Figures 3.10(c) and (d) report a different classifier, generated by fixing the following constraints: each classification stage should classify at least 30% of the available tokens, and should misclassify at most 10% (if possible). The methodology adopted for achieving this, was to perform several tuning trials at each stage, by starting from the value 100% of confidence and progressively lowering it until the criterion is met. Figure 3.10(d) describes the tuning occurred at each classification stage. The constraint over the classification percentage clearly boosts the performance of each single classification stage: as a result, the overall number of classified tokens is 86.7%, with a misclassification rate of 10.2% and 3.1% unclassified tokens.

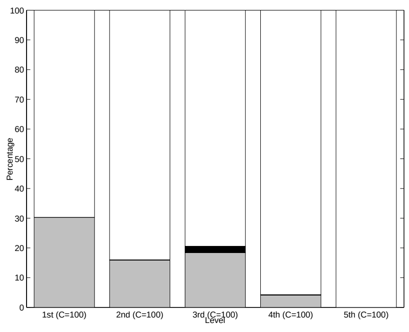
Notice that further effective strategies can be employed, by fixing, e.g., different constraints: in Figure 3.10(e), for example, each classification stage should classify at least 20% of the available tokens, and should misclassify at most 5% of them. Figure 3.11(a), reports a different experiment, where the number of stages is fixed to 4: here, confidence is progressively lowered, and the last stage is tuned to minimize the number of unclassified. Again, Figure 3.11(b) describes the tuning occurred at each stage.

Similar conclusions can be drawn with the other datasets: Figure 3.12, e.g., describes the results on both `BigBook` and `Addresses`. In particular, we adopted three levels (with confidence fixed to 100% in the first two levels) for `BigBook` and four levels (with thresholds 100%, 100%, 85% in the first three levels) for `Addresses`. The bars report the cumulative classification results when different confidence levels are applied in the last classification level.

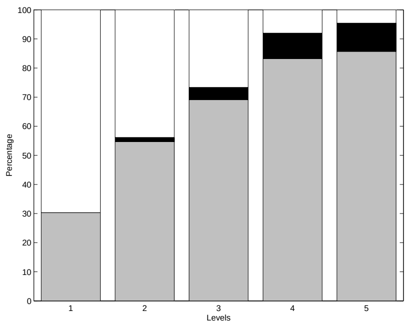
The adoption of multiple classification stages over `BigBook` deserves further discussion about the relation between the size of the labelled data and the number of classification levels which can be defined. Each classification level should build on a separate training set (preprocessed by the preceding levels). Clearly, given a dataset D, the amount of unclassified tokens of D diminishes at subsequent levels. Hence, the size of the training set T_i, required for learn-



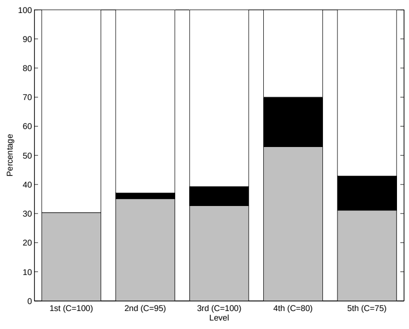
(a) Cumulative performance, 5 levels



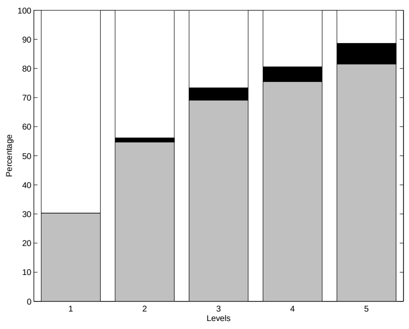
(b) 5 levels, Performance of single stages



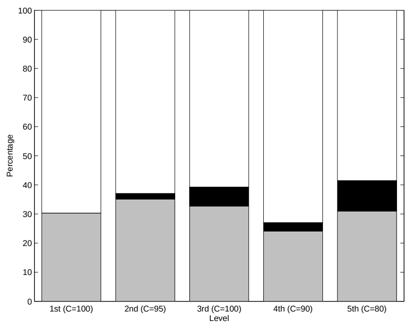
(c) Cumulative performance, 5 levels



(d) 5 levels, Performance of single stages

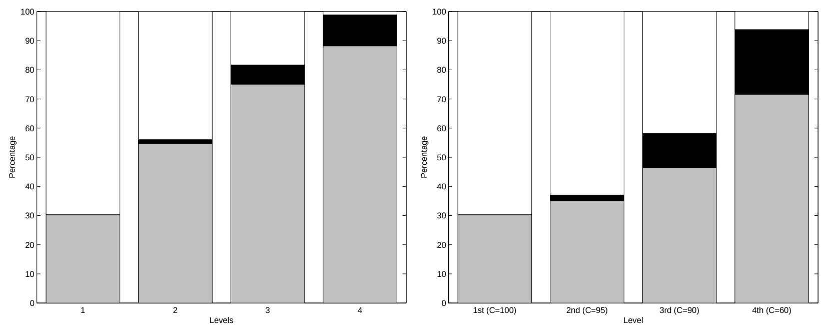


(e) Cumulative performance, 5 levels



(f) 5 levels, Performance of single stages

Fig. 3.10. Effects of five classification levels on `dblp`



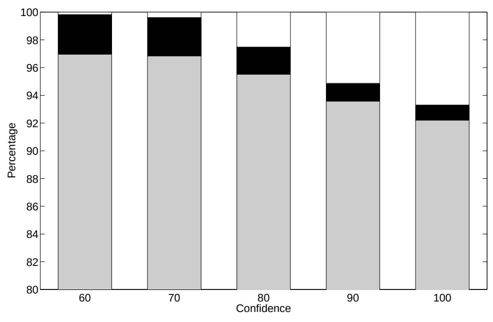
(a) Cumulative performance, 4 levels (b) 4 levels, Performance of single stages

Fig. 3.11. Effects of four classification levels on `dblp`

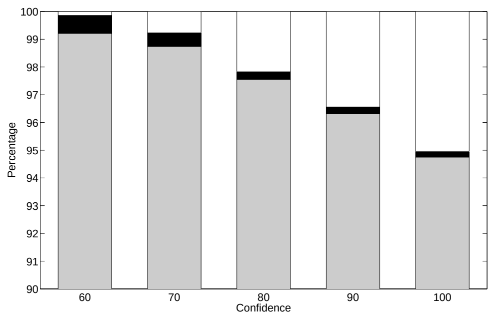
ing rules at level i, should be large enough to guarantee that an adequate number of unclassified tokens are available at that level.

Thus, the size of the training has an influence over the number of classification levels which can be defined: the larger the training set, the higher the number of significant levels. In other words, a small dataset saturates the potential of progressive classification within few levels, and adding further levels does not yield any improvements. This is what happens in the case of the `BigBook` dataset. As already mentioned, the available training set here is quite small. Thus, a classifier exhibiting 100% confidence in the last level, would produce at most 2500 unclassified tokens. This amount would not allow to learn a further meaningful set of rules, since such tokens distribute over different sequences and different attributes.

The conclusion we can draw is that the adoption of multi-stage classification allows to increase recall, by contemporarily controlling the decrease in the overall classification accuracy. The Figures 3.13 and 3.14 show how a proper manipulation of the confidence threshold value over each classification stage allows to achieve this. The contribution of the support threshold is less restrictive for two main reasons: first, it should anyway be kept at very low levels, in order to enable a significant amount of rules; second, small variations are of little significance, and at most at the first level. We here provide details on a set of tests performed on a pipeline of three classifiers, over the `Addresses` datastet. In particular, for brevity sake, we investigate the effects of varying support and confidence for the first-level classifier, whereas the remaining two stages have instead both parameters fixed to respectively 0.5% and 98%. Specifically, in Figures 3.13(a) and 3.13(b) confidence is fixed to 98% and support varies, whereas in Figures 3.14(a) 3.14(b) support is set to 0.5% and confidence varies. Figures 3.13(a) and 3.13(b) show that classification accuracy and recall do not significantly change, especially at higher levels. By the



(a) `Addresses`

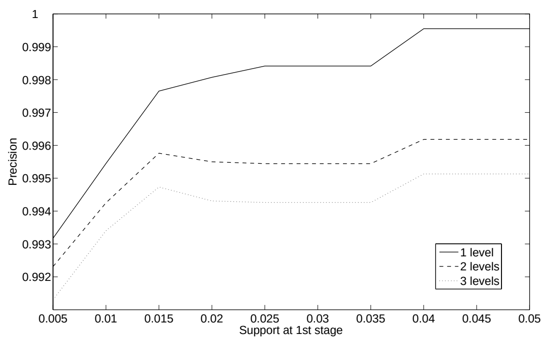


(b) `BigBook`

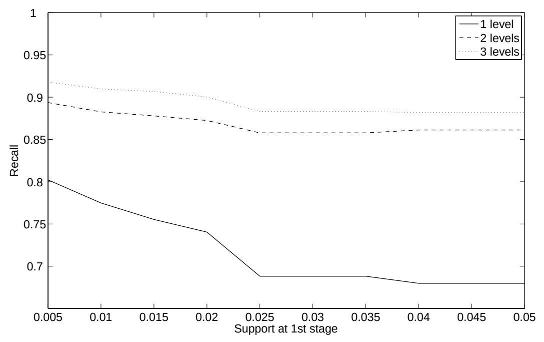
Fig. 3.12. Effects of multiple classification levels

converse, even small variations in the confidence cause significant changes, as testified by Figures 3.14(a) and 3.14(b).

It is interesting to see that, in the above described experiments, the average number of rules which are exploited is nearly stable even on different values of support and confidence (which instead affect the number of discovered rules). Figures 3.15(a) and 3.15(b) depict such a situation. In general, a decrease in support or confidence causes an increase in the overall number of discovered classification rules. However, from experimental evaluations, it emerges that the average number of rules actually applied in the classification process does not significantly vary. This is testified by the bold hatched line in both subfigures, which represents such an average value. As we can see, the number of rules applied fluctuates around 50% of the total number of rules obtained in correspondence of the maximum values of support and confidence.



(a) Accuracy vs. support threshold

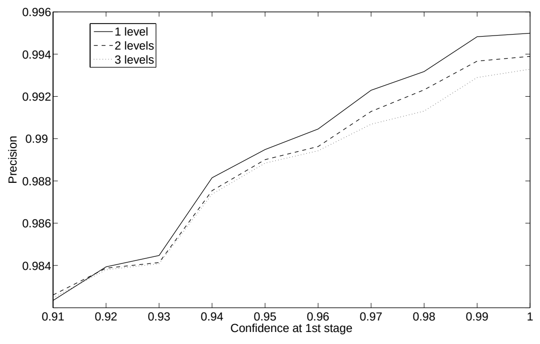


(b) Recall vs. support threshold

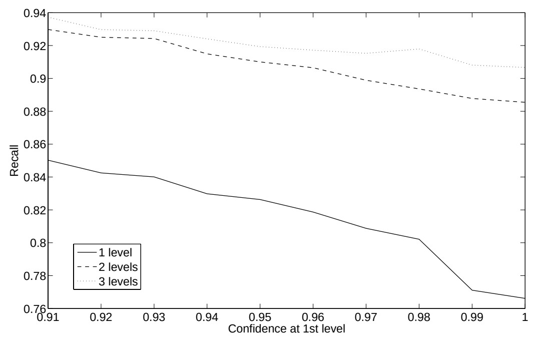
Fig. 3.13. Stability of Precision and Recall with fixed confidence

3.5.4 Comparative Analysis

The exploitation of the "recursive boosting" strategy proposed in this approach is quite new, as it relies on the capability of recovering unclassified tokens in the next stages. To this purpose, the former experiments aimed essentially at checking whether this strategy is effective. In order to asses the practical effectiveness of RecBoost, we here compare the behavior of the RecBoost methodology with consolidated approaches from the literature. To this purpose we preliminarily observe that, although many results are available in the literature, a direct comparison is often difficult, as different data collections and/or different ways of tuning the algorithm parameters have been used. For example, although bibliographic citations extracted from the DBLP database have been extensively used in the literature, the datasets used for the analysis were not made publicly available.



(a) Accuracy vs. confidence threshold

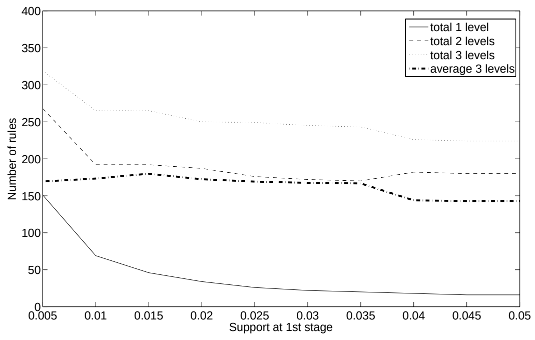


(b) Recall vs. confidence threshold

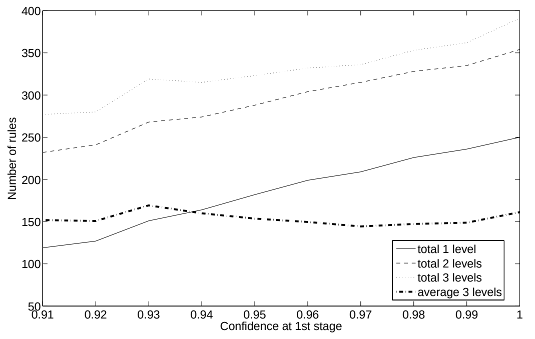
Fig. 3.14. Stability of Precision and Recall with fixed support

In the following we provide a comparison by exploiting the datasets described in the previous sections. We compare our system with the MALLET system [95], which provides the implementation of Conditional Random Fields [88] and with the DATAMOLD system [15]. We refer the reader to Chapter 2 for a detailed description of the techniques underlying such systems. Both MALLET and DATAMOLD are equipped with the same ontology and preprocessing used in RecBoost. In addition, contextual information in the CRFs implemented by MALLET was provided by resorting to the Pre/Post information.

An overall comparison is shown in the graphs of Figure 3.16, which plot the $\mathcal{F}$ values obtained by MALLET, DATAMOLD, and several different instantiation of the RecBoost system. In particular, we consider the classifiers of Figures 3.10, 3.11 and 3.12, and choose, for each dataset, the three instantiations which guarantee the lowest (constrained) value of FN, the lowest (constrained) value of FP, and a "middle" value. The constraint refers to the



(a) Number of rules vs. support

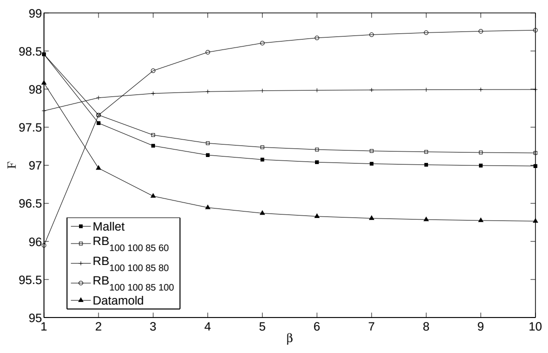


(b) Number of rules vs. confidence

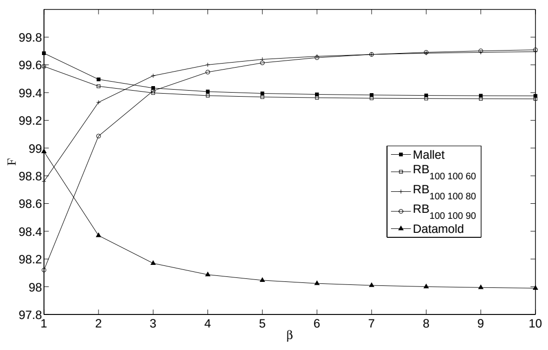
Fig. 3.15. Size of classifiers and average number of rules applied

possibility of maintaining an acceptable value of *TP*. For the `dblp` dataset, we also show an instantiation

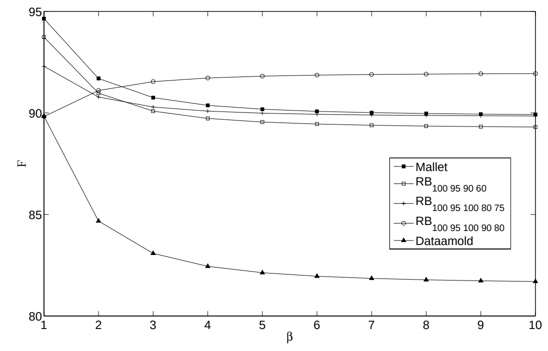
As we can see from the Figure 3.16, the gain in the $\mathcal{F}$ value is evident for $\beta > 2$. Table 3.3 details the results. Here we compare with MALLET, DATAMOLD and the version of RecBoost (RecBoost^1 in the tables), relative to a single stage of classification which achieves the highest value of *TP* in Figure 3.7. MALLET (and in some cases even DATAMOLD) typically achieves a high rate of correctly classified tokens at the expense of a higher misclassification rate. Also, RecBoost^1 may achieve a higher *TP* than the approaches with multiple classification stages. However, the latter exhibit a higher affordability (which is even higher than that of MALLET and DATAMOLD). In practice, the adoption of multiple stages allows to achieve a higher precision, at the expense of a lower recall. Clearly, a proper tuning at the higher levels makes the RecBoost system highly competitive: in `Addresses`, for example,



(a) `Addresses`



(b) `BigBook`



(c) `dblp`

Fig. 3.16. Trends of $\mathcal{F}$-measures compared

the performance of the more conservative classifier (the one which tries to minimize *FN*) is even better than MALLET.

In practice, the recursive boosting offered by progressive classification allows to maintain a higher control over the overall misclassification rate, by forcing stronger rules which, as a side effect, exhibit a higher “locality”. Thus, RecBoost is more reliable in scenarios where misclassifying is worst than avoiding to classify.

Methods	`Addresses`				
	TP	FP	FN	P	R
DATAMOLD	96.23%	3.77%	0%	96.23%	100%
MALLET	96.96%	3.04%	0%	96.96%	100%
RecBoost^1	93,74%	6,24%	0,02%	93,76%	99,98%
RecBoost$^\diamond$	96,96%	2,86%	0,18%	97,14%	99,81%
RecBoost$^+$	95,53%	1,95%	2,52%	98,00%	97,43%
RecBoost$^\bigcirc$	92,21%	1,09%	6,70%	98,83%	93,23%

Methods	`BigBook`				
	TP	FP	FN	P	R
DATAMOLD	97.97%	2.03%	0%	97.97%	100%
MALLET	99,37%	0,63%	0%	99,37%	100%
RecBoost^1	99,01%	0,98%	0,01%	99,02%	99,99%
RecBoost$^\diamond$	99,21%	0,65%	0,14%	99,35%	99,83%
RecBoost$^+$	97.55%	0.28%	2.17%	99.71%	97.82%
RecBoost$^\bigcirc$	96,31%	0,25%	3,44%	99,74%	96,55%

Methods	`dblp`				
	TP	FP	FN	P	R
DATAMOLD	81,55%	18,45%	0%	81,55%	100%
MALLET	89,83%	10,17%	0%	89,83%	100%
RecBoost^1	83,80%	16,20%	0%	83,80%	100%
RecBoost$^\diamond$	88,20%	10,66%	1,14%	89,22%	98,73%
RecBoost$^+$	85,69%	9,74%	4,57%	89,80%	94,94%
RecBoost$^\bigcirc$	81,53%	7,10%	11,37%	91,98%	87,76%

Table 3.3. Comparison against MALLET and DATAMOLD

3.6 Qualitative Comparison

In Chapter 2, we already faced the text segmentation by reviewing several (rule-based or stochastic) approaches dealing with such a problem. Therefore, in this section, we limit ourself just to highlight some differences between RecBoost and some direct challenging approaches.

Amongst these approaches we do not intentionally consider algorithms just relying on HTML separator tags (*wrappers*), since are not effective in domains where data do not necessarily adhere to a fixed schema. Indeed, instances in our problem are more irregular, since the order of fields is not fixed, not all attributes are present, etc. The classification of an item is better performed according to its neighboring words, absolute/relative position in the string, numeric/alphanumeric characters, and so on. To our knowledge, few exceptions are capable of effectively dealing with such features. For instance, WHISK [133] can deal with missing values and permutations of fields, but it requires a "complete" training set, i.e. a set of examples including all the possible occurrences of values.

Conversely, state-of-the-art approaches (e.g., [1, 15, 88, 96, 127]) rely on stochastic models: Hidden Markov Models (HMMs), Maximum Entropy Markov Models (MEMMs) and Conditional Random Fields (CRFs).

In particular, HMMs are widely used models. Schema reconciliation with HMMs can be accomplished by learning the structure of a HMM and applying it to unknown examples. Let just recall (for more details see Section 2.4.1) that, a HMM consists of a set of states and directed edges among such states. Two particular states are the *initial* and the *final* states, where the former has no incoming edges, whereas the latter has no outgoing edges. Every state of the HMM, except from the initial and the final ones, represents a class label and is associated with a dictionary, grouping all the terms in the training set that belong to the class. Edges among states are associated with transition probabilities. A textual sequence can be classified if its constituting terms can be associated to states of the HMM, that form a path between the initial and final states. Precisely, the classification is pursued by associating a single term to all those states, whose corresponding dictionaries include the term. Hence, a sequence of textual terms is mapped to multiple paths throughout the HMM. Transition probabilities are then exploited to identify the most probable path and, hence, to accordingly classify the terms in the sequence at hand. Clearly, those sequence, whose tokens do not form any path between the initial and final states, cannot be classified.

The effectiveness of the approaches based on HMMs strongly depends on the number of distinct terms occurring in the training set. Furthermore, the classification of individual term sequences in one step, i.e., subjected to the existence of corresponding paths throughout the automaton, is a major limitation of HMMs. Indeed, depending on the outcome of the training phase, these cannot undertake the reconciliation process, whenever a path for the sequence at hand does not exist. Also, the existence of one or more paths for a given input sequence may not determine a proper reconciliation.

Worst, HMMs represent "global classification models", since they tend to classify each term of the sequence under consideration, and hence are quite sensitive to unknown tokens.

In other approaches, emphasis have been paid to the analysis of token context (i.e. of the tokens following and preceding the one at hand) for more accurate reconciliation. In particular, MEMMs [96], i.e. conditional models that represent the probability of reaching a state given an observation and the previous state, can be seen as an attempt at contextualizing token reconciliation. However, as discussed in Section 2.4.2, MEMMs suffer from the well known *label-bias problem* [88].

Conditional random fields (CRFs) [88, 95] are another probabilistic model for text labeling and segmentation (see Section 2.4.3). The underlying idea is to define a conditional probability distribution over label sequences, given a particular observation sequence, rather than a joint distribution over both label and observation sequences. CRFs provide two major advantages. First, their conditional nature relaxes the strict independence assumptions required by HMMs to guarantee tractable inference. Second, CRFs avoid the label bias problem.

It is worth noticing that, despite the improvements introduced by MEMMs and CRFs to the HMM technology, they still represent global classification models, since they tend to classify each term into the sequence under consideration, and hence do not prevent the problem of misclassifying unknown tokens.

An attempt towards an unsupervised approach to text reconciliation is proposed by CRAM system in [1] (see Section 2.4.1). The basic idea here is to exploit *reference relations* for building segmentation models. Let recall that, the reference relation denotes a collection of structured tuples that are specific to a domain of interest, and exemplify clean records for that domain.

It is easy to observe that, the exploitation of reference tables is a natural way of automatically building training sets for the text reconciliation problem described beforehand. And indeed, although declared as an unsupervised approach, this technique suffers from two general weaknesses that are inherent of supervised methods. Foremost, a reference relation may not exists for a particular applicative scenario. Also, whenever the overall number of tuples involved is not sufficiently large, the columns of the employed relations may not adequately rich dictionaries of basic domain tokens. This would affect the overall segmentation effectiveness.

As to a more specific comparison with our contribution, the reference table approach requires to initially learn the order with which attributes appear within the input data. By contrast, though being a supervised approach, RecBoost does not rely on learning attribute order from training data. This is due to the adoption of classification rules, that allow the reconciliation of a given token on the sole basis of the relationships among the entities (i.e. further textual tokens, ontological categories and attributes) in the context surrounding the token at hand. Moreover, segmentation with respect to a given attribute order relies on the underlying assumption that such an ordering is fixed across input sequences. This may make reconciliation problematic when, instead, the tokens of two or more attribute values are interleaved (rather than being concatenated) in the data to segment.

Finally, the reference table approach adopts basically HMMs (the so called ARMs) for reconciliating individual attribute values, and hence suffer from its aforementioned limitations. Roughly speaking, ARMs are global classification models and, hence, overly specific in attribute recognition as far as three aspects are concerned, namely positional, sequential and token specificity. These aspects impose suitable generalizations for the ARMs: the adoption of a fixed three-layered topology capable of dealing with positional and sequential specificities and the exploitation of token hierarchies for mitigating token specificity. On the contrary, RecBoost relies on association rules for attribute value reconciliations. Association rules are better suited at detecting local patterns, especially when the underlying data to segment contain many contrasting specificities. Moreover, a natural generalization of classifiers, i.e., the improvement of their classification accuracy, is trivially obtained by attempting to reduce classifier complexity, via attribute and rule pruning.

3.7 Conclusions

In this chapter we presented RecBoost, a novel approach to schema reconciliation, that fragments free text into tuples of a relational structure with a specified attribute schema. Within RecBoost, the most salient features are the combination of ontology-based generalization with rule-based classification for more accurate reconciliation, and the adoption of *progressive classification*, as a major avenue towards exhaustive text reconciliation. An intensive experimental evaluation on real-world data confirms the effectiveness of our approach.

There are some directions that are worth further research. First, notice that the proposed methodology is, in some sense, independent from the underlying rule-generation strategy. In this respect, it is interesting to investigate the adoption of alternative strategies for learning local classification models. This line is also correlated with the effort for identifying a fully-automated technique for setting the parameters of *progressive classification*, in terms of required classification stages. Since parameters are model-dependent, two alternate strategies can be either to investigate different, parameter-free models, or to detect ways to enable a natural way of fixing the parameters of the system, on the basis of the inherent features of the text at hand, rather than relying on pre-specified estimates. The experimental section already contains some pointers in the latter direction: however, more robust methods need in-depth investigation.

In addition, interesting would be investigate the development of an unsupervised approach to the induction of an attribute descriptor from a free text. This would still allow reconciliation, even in the absence of any actual knowledge about the textual information at hand.

4

Approaches to Data Reconciliation

4.1 Introduction

In this chapter, we focus on surveying some well-known techniques to solve the problem of *Data Reconciliation* (or *lexical heterogeneity*) regarding tuples in a single o multiple databases. This problem occurs when tuples have identically structured fields across databases, but the data use different representations to refer to the same real-world object (e.g., *StreetAddress=75 S. 5th Avn.* vs. *StreetAddress=75 South Fifth Avenue*). The goal of data reconciliation is to identify records in the same or different databases that refer to the same real world entity, even if the records are not identical.

More in detail, we focus on the case where the input is a set of *structured* and *properly segmented* records (database records). Hence, in this chapter, we do not cover solutions for other problems such as *anaphora resolution* [102], in which the problem is to locate different mentions of the same entity in *free text*.

This chapter is organized as follows. In Section 4.2, we briefly discuss the necessary steps in the data de-duplication process, before the duplicate record detection phase starts. Then, Section 4.3 describes techniques used to match individual fields, and Section 4.4 presents techniques for matching records containing multiple fields. Section 4.5 describes methods for improving the efficiency of the duplicate record detection process, and Section 4.6 presents some commercial tools used in industry for duplicate detection purpose. Finally, Section 4.7 concludes the chapter.

4.2 Data Preparation

Duplicate record detection is the process of identifying different or multiple records that refer to a unique real world entity. In the product space of two tables, a *match* is a pair that represents the same entity and a *non-match* is a pair that represents two different entities. Within a single table, a *duplicate* is a record that represents the same entity as another record in the same database. Common record identifiers such as names, addresses and code numbers (e.g., SSN, object identifier) can be used to identify matches. The final goal in record matching is to determine whether a comparison record corresponds to a matched or a non-matched pair of database records. Unfortunately, the presence of errors in the identifying information makes this problem hard.

Typically, the process of duplicate detection is preceded by a *data preparation stage*. In this phase, data entries are stored in a uniform manner in the database, resolving (at least partially) the structural heterogeneity problem. The data preparation stage includes a *parsing*, a *data transformation*, and a *standardization* step. The approaches that deal with data preparation are also described under the "umbrella name" *ETL* (Extraction, Transformation, Loading). These steps improve the quality of the in-flow data and make the data comparable and more usable. A complete collection of papers related to various data transformation approaches can be found in [123]. More in detail:

- ***Parsing*** or ***Segmentation*** (according to the terminology used in the previous chapters) is the first critical component in the data preparation stage. Segmentation locates, identifies and isolates individual data elements in the source files. Segmentation makes easier to correct, standardize and match data because it allows the comparison of individual components, rather than of long complex string of data. For example, the appropriate segmentation of name and address component into consistent packets of information is a crucial part in the record matching process. Several segmentation methods have been proposed in literature (e.g., [96, 15, 1, 88]) and we reviewed them in Chapter 2.
- ***Data Transformation*** refers to simple conversions that can be applied to the data in order to conform them to the data types of their corresponding domains. In other words, this type of conversion focuses on manipulating one field at a time, without taking into account the values in related fields. The most common form of data transformation is the conversion of a data element from one data type to another. Renaming a field from one name to another is considered data transformation as well. Encoded values in external data is another problem addressed at this stage. This conversion is relevant in order to consent that records from different sources can be compared in a uniform manner. Range checking is yet another kind of data transformation which ensures that data in a field falls within an expected range, usually a numeric or date range. Finally, dependency checking compares the value in a particular field to the values in another field in order to ensure a minimal level of consistency in the data.
- ***Data Standardization*** refers to the process of standardizing the information represented in certain fields to a specific content format. Without standardization, many duplicate entries could erroneously be designated as non-duplicates, based on the fact that common identifying information

cannot be compared. One of the most common standardization applications involve address information. Indeed, there is no one standardized way to capture addresses: the same address can be represented in many different ways. Data and time formatting and name and title formatting pose other standardization difficulties in a database. Data standardization is a rather inexpensive but a very important step in the de-duplication process because it can lead to fast identification of duplicates. For example, if the only difference between two records is the differently recorded address (*55 West Fifth Street* vs. *55 W 5th St.*), then the data standardization step would make the two records identical, alleviating the need for more expensive approximate matching approaches, that we describe further in this chapter.

After the data preparation phase, data are typically stored in tables having comparable fields. The next step is to identify which fields should be compared. For example, it would not be meaningful to compare the contents of the field *FirstName* with the field *Address*. In [111] a supervised technique for understanding the "semantics" of the fields contained in web databases is presented. Moreover, in [41] this concept is significantly extended and a "signature" from each field in the database is extracted. This signature summarizes the content of each column in the database. Then, the signatures are used to identify fields with similar values, fields whose contents are subsets of other fields and so on.

Despite parsing, data standardization and identification of similar fields, it is not trivial to match duplicate records. Misspellings and different conventions for recording the same information still result in different, multiple representations of a unique object in the database.

4.3 Field Matching and String Matching Techniques

One of the most common sources of mismatches in database entries is the typographical variations of string data. Therefore, duplicate detection typically relies on string comparison techniques to deal with typographical variations. Multiple methods have been developed for this task, and each method works well for particular types of errors. Instead, if errors appear in numeric fields, the related research is still in its initial phase. In this section we review some techniques that have been applied for matching string fields. Next, we briefly introduce some common approaches to deal with errors in numeric data.

4.3.1 Character-based similarity metrics

The character-based similarity metrics are designed to deal with typographical errors. The following similarity metrics we cover here:

- Edit distance.

- Smith-Waterman distance.
- Affine gap distance.
- Jaro distance metric.
- Q-gram distance.

Edit distance

The *Edit distance* between two string s and t is the minimum number of *edit operations* that convert s to t. There are four types of edit operations:

- *Copy* the next letter in s to the next position in t.
- *Insert* a new letter in t that does not appear in s.
- *Substitute* a different letter in t for the next letter in s.
- *Delete* the next letter in s; that is, don't copy it to t.

In the simplest form of edit distance, the copy operation has cost zero, whereas all other operations have cost one. This version of edit distance is also referred in literature as *Levenshtein distance* [90]. In order to understand how the edit distance is computed, let consider mapping the string s = "Willlaim" to t = "William" using the edit operations above introduced. Table 4.1 shows one possible sequence of these operations (the vertical bar represents a "cursor" in s or t indicating the next letter).

s	***t***	***Operation***
|Willlaim	|	
W|illlaim	W|	Copy "W"
Wi|lllaim	Wi|	Copy "i"
Wil|llaim	Wil|	Copy "l"
Will|laim	Will|	Copy "l"
Willl|aim	Will|	Delete "l"
Willla|im	Willi|	Substitute "i" for "a"
Willlai|m	Willia|	Substitute "a" for "i"
Willlaim|	William|	Copy "m"

Table 4.1. Example of an edit-distance computation

According to the costs of edit operations in the Levenshtein version of edit distance, this is the least expensive sequence for s and t. Then, the edit distance between "Willlaim" and "William" is 3.

A fairly efficient scheme exists for computing the lowest-cost edit sequence for these operations. The trick consists in considering a slightly more complex function $D(s, t, i, j)$ which is the edit distance between the first i letters in s and the first j letters in t. Let s_i denote the i-th letter of s, and similarly, let t_j be the j-th letter of t. Then $D(s, t, i, j)$ can easily recursively defined, where $D(s, t, 0, 0) = 0$ and

$$D(s,t,i,j) = min \begin{cases} D(s,t,i-1,j-1) & \text{if } s_i = t_j\text{, and you copy } s_i \text{ to } t_j \\ D(s,t,i-1,j-1)+1 & \text{if you substitute } t_j \text{ for } s_j \\ D(s,t,i,j-1) & \text{if you insert the letter } t_j \\ D(s,t,i-1,j) & \text{if you delete the letter } s_i \end{cases} \tag{4.1}$$

This recursive definition can be efficiently evaluated by using dynamic programming techniques [105]. Specifically, for a fixed s and t, the $D(s,t,i,j)$ values can be stored in a matrix that is filled in a particular order. The total computation effort for $D(s,t,|s|,|t|)$ (the edit distance between s and t) is approximately $O(|s| \cdot |t|)$. Landau and Vishkin [89] presented an algorithm for detecting in $O(max\{|s|,|t|\} \cdot k)$ whether two strings s and t have edit distance less than k. To this purpose, notice that if $||s| - |t|| > k$, then, by definition, the two strings do not match within distance k, so:

$$O(max\{|s|,|t|\} \cdot k) \sim O(|s| \cdot k) \sim O(|t| \cdot k)$$

Edit distance metrics are widely used, not only for text processing but also for biological sequence alignment, and many variations are possible. The *Needleman-Wunsch distance* [106] is a natural extension to *Levenstein distance* that introduces additional parameters defining each possible character substitution's cost and the cost of insertions and deletions. For instance, the cost of replacing O with 0 might be chosen smaller than the cost of replacing f with q. This variation can be simply implemented by modifying Equation (4.1) replacing the second term of the min with something such as $D(s,t,i-1,j-1) + substitutionCost(s_i, t_j)$. Ristad and Yiannilos [122] presented a method for automatically determining such costs from a set of equivalent words that are written in different ways.

The edit distance metrics work well for catching typographical errors, but they are typically ineffective for other types of mismatches.

Smith-Waterman distance

Smith and Waterman [132] described an extension of edit distance in which mismatches at the beginning and the end of strings have lower costs than mismatches in the middle. Therefore, this distance metric works well for matching strings that have been truncated or shortened (e.g., *"Stanford U."* vs. *"Stanford University"*). The distance between two strings can be computed using a dynamic programming technique, based on the *Needleman and Wunsch* algorithm [106], which seeks to locate the best alignment between the two strings.

Pinheiro and Sun [114] proposed an analogue similarity measure, which tries to find the best character alignment for two compared strings, so that the number of character mismatches is minimized. For instance, the strings $s = ABcDeFgF$ and $t = AxByDzFH$ can be aligned as in Figure 4.1.

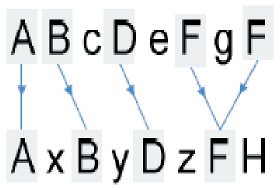


Fig. 4.1. The Pinheiro and Sun similarity metric alignment

This metric is similar to the *Smith-Waterman distance* and to the *Affine gap distance*, which next will be discussed.

Affine gap distance

While the *Smith-Waterman distance* modifies the Levenshtein metric in order to discount mismatching text at the beginning and at the ending of strings, it places stronger penalties on mismatches in the middle. When the errors are in the middle (e.g., *"John R. Smith"* vs. *"Johnathan Richard Smith"*) this can create a problem. The *Affine gap distance* offers a solution to this issue by introducing two extra edit operations:

- *Open gap* that is the cost for inserting the first character.
- *Extend gap* that is the cost for inserting additional characters.

The cost of extending the gap is usually smaller than the cost of opening a gap. This results in smaller cost penalties for gap mismatches than the equivalent cost under the edit distance metric. Bilenko et al. [12], similarly to what Ristad and Yiannilos [122] proposed for edit distance, describe how to train an edit distance model with affine gaps.

Jaro distance metric

Jaro [78] introduced a string comparison algorithm mainly used for comparing last and first names. Essentially, this metric is based on the number and order of common characters between two strings. The basic algorithm for computing the Jaro metric for two string s and t includes the following steps:

1. Compute the string lengths $|s|$ and $|t|$.
2. Find *common* characters c in the two strings. By definition, a character a_i in s is "in common" with t iff there is a $b_j = a_i$ in t such that $i - H \leq j \leq i + H$, where $H = min\{|s|, |t|\}/2$.
3. Find the number of *transposition* k. The number of transpositions is computed as follows: the i-th common character in s is compared with the i-th common character in t and each non-matching character is a transposition.

Then, the Jaro metric for s and t is:

$$Jaro(s,t) = \frac{1}{3} \cdot \left(\frac{c}{|s|} + \frac{c}{|t|} + \frac{c - k/2}{c} \right)$$

To better understand the intuition behind this metric, consider the matrix $\mathbf{M}$ in Table 4.2, which compares the string s="WILLLAIM" and t="WILLIAM".

	W	I	L	L	I	A	M
W	$\boxed{1}$	**0**	**0**	0	0	0	0
I	**0**	$\boxed{1}$	**0**	**0**	1	0	0
L	**0**	**0**	$\boxed{1}$	**1**	**0**	0	0
L	0	**0**	**1**	$\boxed{1}$	**0**	**0**	0
L	0	0	**1**	**1**	$\boxed{0}$	**0**	**0**
A	0	0	0	**0**	**0**	$\boxed{1}$	**0**
I	0	1	0	0	**1**	**0**	$\boxed{0}$
M	0	0	0	0	0	**0**	**1**

Table 4.2. The Jaro metric

The boxed entries are the main diagonal, and $\mathbf{M}(i,j) = 1$ if and only if the i-th character of s equals the j-th character of t. As discussed above, the Jaro metric is based on the number of characters in s that are in common with t. Within $\mathbf{M}$, the i-th character of s is in common with t if $\mathbf{M}(i,j) = 1$ for some entry (i,j) that is "sufficiently close" to the main diagonal of $\mathbf{M}$. Sufficiently close means that $|i-j| < min\{|s|,|t|\}/2$ (shown in bold in the matrix). In Table 4.2, it is easy to see that the number of characters in common c are 7 and the number of transpositions k are 2. Then, $Jaro(\text{“}WILLLAIM\text{”}, \text{“}WILLIAM\text{”}) = 0.333 \cdot (\frac{7}{8} + 1 + \frac{6}{7}) = 0.910$

Winkler and Thibaudeau [154] proposed a variant of the Jaro metric that also uses the length P of the longest common prefix of s and t. Letting $P' = max(P, 4)$, then

$$Jaro - Winkler(s,t) = Jaro(s,t) + (P'/10) \cdot (1 - Jaro(s,t))$$

This emphasizes matches in the first few characters, since prefix matches are generally more important for surname matching.

Q-gram distance

The *q-grams* are short character substrings of length q of the database strings [146, 145]. Letter q-grams, including trigrams, bigrams, and/or unigrams, have been used in a variety of ways in text recognition and spelling correction [86]. One natural extension of q-grams are the *positional q-grams* [138], which also record the position of the q-gram in the string. The q-grams approach is widely used in the string matching scenario. In particular, Gravano et al. [65, 64] showed how to use positional q-grams to locate efficiently similar strings within a relational database.

Given a string s, its positional q-grams are obtained by "sliding" a window of length q over the characters of s. Since q-grams at the beginning and the end of the string can have fewer than q characters from s, the strings are conceptually extended. More in detail, the string s is padded by prefixing or suffixing it with $q-1$ occurrences of a special padding character (e.g., # and *) not included in the original alphabet. Thus, each q-gram contains exactly q characters, though some of these may not be from the original alphabet.

The main intuition behind the use of q-grams as a foundation for approximate string matching is that, when two strings s and t are similar (e.g., are within a small edit distance of each other), they share a large number of q-grams in common [138, 145]. The following example illustrate this observation.

Consider the string s=`john_smith` and the string t=`john_a_smith`. The positional q-grams of length q=3 for the string s are {(1,##j), (2,#jo), (3,joh), (4,ohn), (5,hn_), (6,n_s), (7,_sm), (8,smi), (9,mit), (10,ith), (11,th*), (12,h**)}. Similarly, the positional q-grams of length q=3 for the string t (which is at an edit distance of two from s) are {(1,##j), (2,#jo), (3,joh), (4,ohn), (5,hn_), (6,n_a), (7,_a_), (8,a_s), (9,_sm), (10,smi), (11,mit), (12,ith), (13,th*), (14, h**)}. By ignoring the position information, the two q-gram sets have 11 q-grams in common. Interestingly, only the first five positional q-grams of the string s are also positional q-grams of the string t. However, an additional six positional q-grams in the two strings differ in their position by just two positions. This illustrates that, in general, the use of positional q-grams for approximate string matching involves comparing position of "matching" q-grams within a certain "band".

4.3.2 Token-based similarity metrics

As discussed above, character-based similarity metrics works well for typographical errors. However, it is often the case that typographical conventions lead to rearrangement of words. For instance, the strings "John Smith" and "Smith, John" are likely to be duplicates, even if they aren't close in edit distance. Then, it is clear that, in such cases, character-level metrics fail to capture the similarity of the entities. Token-based metric try to compensate for this problem. In this approach, two string s and t are previously converted in token multisets S and T (where each token is a word) and then some metrics on these multisets are applied.

Jaccard similarity

One simple and often effective token-based metric is the *Jaccard similarity.* By considering two string s and t, if S and T are respectively their word sets, the Jaccard similarity is simply defined as

$$Jaccard(s,t) = \frac{|S \cap T|}{|S \cup T|}$$

For instance, consider the strings s= *"Comput. Sci. Dept. California University San Diego"* and t= *"Department Computer Science Univ. Calif. San Diego"*. It is easy to see that the size of intersection (number of matching words) between S and T is 2, whereas the size of union (number of distinct words) between S and T is 12. Then, the $Jaccard(s,t) = 0.166$.

Atomic Strings

Monge and Elkan [103] proposed a basic algorithm for matching text fields based on *atomic strings*. By definition, an atomic string is a sequence of alphanumeric characters delimited by punctuation characters. Two atomic strings match if they are the same string or if one is a prefix of the other. A simple definition of the similarity degree of two strings is the number of their matching atomic strings divided by their average number of atomic strings.

For instance, consider the strings s= *"Comput. Sci. & Eng. Dept., University of California, San Diego"* and t= *"Department of Computer Science, Univ. Calif., San Diego"*. After removing stop words (e.g., and, in, for, the, of, on, &), $k = 6$ atomic strings in s match atomic strings in t, namely *Comput., Sci., San, Diego, Univ., Calif.*. Then, the overall matching score is $k/(|S| + |T|)/2) = 0.8$.

TF-IDF

The *term frequency-inverse document frequency* or *cosine similarity*, which the IR community widely uses, is defined as:

$$TF - IDF(S,T) = \sum_{w \in S \cap T} V(w,S) \cdot V(w,T)$$

If $TF_{w,S}$ is the frequency of word w in S and IDF_w is the inverse of the fraction of names in the corpus that contain w, then:

$$V'(w,S) = log(TF_{w,S} + 1) \cdot log(IDF_w)$$

and

$$V(w,S) = \frac{V'(w,S)}{\sqrt{\sum_{w'} V'(w,S)^2}}$$

where the sum is over the entire set of words.

Cohen [33] described a system named *WHIRL*, that adopts the cosine similarity combined with tf-idf weighting scheme to compute the similarity of two strings.

The cosine similarity metrics works well for a large variety of entries, and is insensitive to the location of words, thus allowing natural word moves and swaps (e.g., “John Smith” is equivalent to “Smith, John”). Also, introduction of frequent words affects only minimally the similarity of the two strings due to the low *IDF* weight of frequent words. Unfortunately, this similarity metric does not capture word spelling errors, especially if they are pervasive and affect many of the words in the strings. For example, the corrupted strings “Compter Science Departement” and “Deprtment of Computer Science” will have similarity zero under this metric.

To solve this problem, Bilenko et al. [12] suggest to combine token-based and character-based methods in a hybrid approach named *SoftTF-IDF*. In the SoftTF-IDF metric, the similarity is affected not only by tokens that appear both in S and T but also by token in S such that a “similar” token (according to some metrics) appears in T. More in detail, let sim be a secondary similarity function that performs well on short strings (e.g., Jaro-Winkler). Let $CLOSE(\theta, S, T)$ be the set of words $w \in S$ such that some $v \in T$ exists for which $sim(w, v) > \theta$. Moreover, for $w \in CLOSE(\theta, S, T)$, let $N = \max_{v \in T} sim(w, v)$, then

$$\mathit{SoftTF} - \mathit{IDF}(S, T) = \sum_{w \in CLOSE(\theta,S,T)} V(w, S) \cdot V(w, T) \cdot N(w, T)$$

Gravano et al. [66] extended the WHIRL system to handle spelling errors by using tf-idf similarity on q-grams rather on words. Indeed, in this setting, a spelling error minimally affects the set of common q-grams of two strings. Then, the strings “Gteway Communications” and “Comunications Gateway” have higher similarity under this metric, despite the swap and the spelling errors in both words.

4.3.3 Numeric Similarity Metrics

So far, we have dealt only with string attributes. However, there are many data types that are commonly encountered in practice. Numeric data is of particular interest. The conventional approach of finding fuzzy matches with respect to a given numeric value is to issue a range query against the corresponding field. But this non take advantage of the data distribution to return a better result [85].

Again, many search engines work with just the string representation of numeric values. This approach is inadequate for flexible matching purposes. For instance, by trying to perform a google search on “186000”, the engine returns few pages mentioning the speed of light, but a search on “185900” does not find any such pages. Part of the reason is that the string representation of numbers which are very close may not have enough tokens in common. To this purpose, the extension of cosine similarity metrics to non-string data type is a very interesting research direction [3].

4.4 Detecting Duplicate Records

In the previous section, we surveyed some approaches that can be used to match individual fields of a record. However, in most real-life situations, the records consist of multiple fields, making the duplicate detection problem much more complicated. In this section, we review methods that are used for matching records with multiple fields. The presented methods can be broadly divided into two categories:

- Approaches that rely on training data to "learn" how to match the records. This category includes some probabilistic approaches and supervised machine learning techniques.
- Approaches that rely on domain knowledge or on generic distance metrics to match records. This category includes approaches that use declarative languages for matching, and approaches that devise appropriate distance metrics for duplicate detection task.

The section is organized as follows: initially, in Section 4.4.1 we present the probabilistic approaches for solving the duplicate detection problem. In Section 4.4.2 we survey approaches using supervised machine learning techniques, and in Section 4.4.3 we describe variations based on active learning methods. Section 4.4.4 shows distance-based techniques, and Section 4.4.5 describes declarative approaches to the duplicate record detection. Finally, Section 4.4.6 covers unsupervised learning techniques.

4.4.1 Probabilistic Matching Models

Firstly, let introduce some helpful concepts and the notation used to treat the probabilistic models.

Notation

Let A and B denote the tables that we want to match and let assume, without loss of generality, that A and B have n comparable fields. In the duplicate detection problem, each tuple pair $\langle \alpha, \beta \rangle$ where $\alpha \in A$ and $\beta \in B$, is assigned to one of two classes M and U. The class M contains record pairs that represent the same entity ("*match*") and the class U contains the record pairs that represent two different entities("*non-match*").

Many matching problems are more constrained than this statement of the problem. For instance, if each record in data source B refers a distinct entity, a record in data source A cannot be matched to two records at the same time in data source B. Cohen called this the *constrained matching problem* [37]. It is more generally referred to as *1-1 linkage* in comparison to the alternative *1-many linkage*. 1-1 linkage, since it has more constraints, is a harder optimization problem [37].

The generic j-th (with $j = 1 \ldots |A \times B|$) tuple pair $\langle \alpha, \beta \rangle$ can be represented as a random vector $\underline{x}^j = [x_1, ..., x_n]^T$ in X, where X denotes the space of all possible comparison vectors and T denotes the transpose of the vector. The n components in $\underline{\mathrm{x}}$ corresponds to the n comparable fields of A and B. Each x_i^j shows the level of agreement of the i-th field for the j-th record pair. Many approaches use binary values for the x_i's and set $x_i = 1$ if the field i agrees and set $x_i = 0$ otherwise. In order to have a less heavy notation, in the following we often drop the superscript from the random vector $\underline{x}$ when its sense is clear from the context.

A random vector may be characterized by a *distribution probability function* $P(\underline{x})$ or by a *density function* $p(\underline{x})$. In the record matching problem we deal with random vectors drawn from the two classes M and U, each of which is characterized by its own density function. This density function is called *conditional density* and is expressed as $p(\underline{x}|M)$ and $p(\underline{x}|U)$ for the classes M and U respectively. The *a-priori probability* is denoted with π_M and π_U for the classes at hand. Since there are only two classes, the following equality holds $\pi_M + \pi_U = 1$. The unconditional density function of a comparison vector $\underline{x}$, sometimes called *mixture density function*, is given by $p(\underline{x}) = \pi_M \cdot p(\underline{x}|M) + \pi_U \cdot p(\underline{x}|U)$. Finally, the *a-posteriori probabilities* are expressed as $p(M|\underline{x})$ and $p(U|\underline{x})$, and can be computed by using the Bayes theorem.

Newcombe et al. [109] were the first to formalize duplicate detection as a Bayesian inference problem. Essentially, the comparison vector $\underline{x}$ is the input to a decision rule that assign $\underline{x}$ to U or M. The main assumption is that $\underline{x}$ is a random vector whose conditional density functions and the a-priori probabilities are assumed to be known.

In the following, we will discuss various techniques that have been developed for addressing this (general) decision problem.

The Bayes Decision Rule for Minimum Error

Let $\underline{\mathrm{x}}$ be a comparison vector, randomly drawn from the comparison space that corresponds to the pair $\langle \alpha, \beta \rangle$. The goal is to determine whether $\langle \alpha, \beta \rangle \in M$ or $\langle \alpha, \beta \rangle \in U$. To this purpose, the decision rule, based simply on probabilities, can be written as follows:

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } p(M|\underline{x}) \geq p(U|\underline{x}) \\ U & \text{otherwise} \end{cases} \tag{4.2}$$

This decision rule indicates that if the probability of the match class M is larger than the probability of the non-match class U, then $\underline{\mathrm{x}}$ is classified as M, and viceversa. The Bayes theorem postulates that:

$$p(C|\underline{x}) = \frac{\pi_C \cdot p(\underline{x}|C)}{p(\underline{x})}$$

where C can take on the values of M or U. By using it, equation (4.2) can be expressed as

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } l(\underline{x}) = \frac{p(\underline{x}|M)}{p(\underline{x}|U)} \geq \frac{\pi_U}{\pi_M} \\ U & \text{otherwise} \end{cases} \tag{4.3}$$

The ratio:

$$l(\underline{x}) = \frac{p(\underline{x}|M)}{p(\underline{x}|U)} \tag{4.4}$$

is called *likelihood ratio.* The term $\frac{\pi_U}{\pi_M}$ denotes the threshold value of the likelihood ratio for the decision. The decision rule that is described in (4.3) is called *Bayes test for minimum error.*

In general, any decision rule which is based on probabilities, does not lead to perfect classification. In order to evaluate the "performance" of a decision rule, the probability of error (the probability for a sample to be assigned to a wrong class) must be calculated. It can be easily shown [72] that the Bayes test results in the smallest probability error, and thus it is an optimal classifier.

As discussed above, this method holds only when the conditional densities and the a-priori probabilities are known. Unfortunately, this is a very rare case. To this purpose, a common way consists in resorting to the *Naïve Bayes* approach. It allows to compute the conditional densities on the base of a conditional independence assumption. In practice, the probabilities $p(x_i|M)$ and $p(x_j|M)$ are independent if $i \neq j$ (similarly for $p(x_i|U)$ and $p(x_j|U)$). In that case, trivially we have:

$$p(\underline{x}|M) = \prod_{i=1}^{n} p(x_i|M)$$
$$p(\underline{x}|U) = \prod_{i=1}^{n} p(x_i|U)$$

The values of $p(x_i|M)$ and $p(x_i|U)$, the so-called *marginal probabilities*, can be easily estimated by using a training set of pre-labeled record pairs. However, the probabilistic model can also be used without resorting to training data. For instance, Jaro [79] suggested using an *Expectation-Maximization* (EM) algorithm [44] to estimate this conditional probabilities.

The Bayes Decision Rule for Minimum Cost

Often, in practice, the minimization of the probability of error, is not the best criterion for creating decision rules, as the misclassifications of M and U may have different consequences. Therefore, it is appropriate to assign a cost c_{ij} to each situation, which is the cost of deciding that $\underline{x}$ belongs to the class i

when $\underline{x}$ actually belongs to the class j. Then, the expected costs $r_M(\underline{x})$ and $r_U(\underline{x})$ of deciding that $\underline{x}$ belongs to the class M and U respectively, are:

$$\begin{aligned} r_M(\underline{x}) &= c_{MM} \cdot p(M|\underline{x}) + c_{MU} \cdot p(U|\underline{x}) \\ r_U(\underline{x}) &= c_{UM} \cdot p(M|\underline{x}) + c_{UU} \cdot p(U|\underline{x}) \end{aligned}$$

In that case, the decision rule for assigning $\underline{x}$ to M becomes:

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } r_M(\underline{x}) < r_U(\underline{x}) \\ U & \text{otherwise} \end{cases} \tag{4.5}$$

It can be easily proved [46], that the minimum cost decision rule for the problem at hand, can be stated as:

$$\langle \alpha, \beta \rangle \in \begin{cases} M & \text{if } l(\underline{x}) > \frac{(c_{MU} - c_{UU}) \cdot p(U)}{(c_{UM} - c_{MM}) \cdot p(M)} \\ U & \text{otherwise} \end{cases} \tag{4.6}$$

Comparing the minimum error and the minimum cost decision rule, we notice that the two decision rules become the same for the special setting of the cost functions to $c_{UM} - c_{MM} = c_{MU} - c_{UU}$. In this case, the cost functions are termed symmetrical. For a symmetrical cost function, the cost becomes the probability of error and the Bayes test for minimum cost specifically addresses and minimizes this error. Different cost functions are generally used when a wrong decision for one class is more critical than a wrong decision for the other class.

Fellegi-Sunter Model (Decision with a Reject Region)

Using the Bayes decision rule when the distribution parameters are known, leads to optimal results. However, even in this ideal scenario, when the likelihood ratio $l(\underline{x})$ is close to the threshold, the error (or cost) of any decision is high [46]. Based on this well-known and general idea on decision theory, Fellegi and Sunter [50], suggested adding an extra "reject" class in addition to the class M and U. This reject class contains record pairs for which it is not possible to make any definite inference, and a "clerical review" is necessary. In practice, these pairs need to be examined manually by experts to decide whether they are true matches or not.

Fellegi and Sunter, making rigorous concepts introduced by Newcombe [109], formalized this idea by defining a *linkage rule* that labels the pair into the comparison space as:

- *designated matches* or links (in set A_1).
- *designated potential matches* or potential links (in set A_2).
- *designated non-matches* or non-links (in set A_3).

For the seek of simplicity, let $m(\underline{x})$ indicate the conditional probability of observing the comparison vector $\underline{x}$ in X given that the record pair $\langle \alpha, \beta \rangle$ is a true match. That is:

$$m(\underline{x}) = p(\underline{x}|M)$$

Similarly, let:

$$u(\underline{x}) = p(\underline{x}|U)$$

denote the conditional probability of observing $\underline{x}$ given that the record pair is a true non-match.

There are two kinds of possible misclassification errors: false matches or *Type I error* (the linkage rule places $\langle \alpha, \beta \rangle \in M$ in A_3), and false non-matches or *Type II error* (the linkage rule places $\langle \alpha, \beta \rangle \in U$ in A_1). The probability of a false match can be written as:

$$p(A_1|U) = \sum_{\underline{x} \in X} u(\underline{x}) \cdot p(A_1|\underline{x})$$

and a probability of a false non-match:

$$p(A_3|M) = \sum_{\underline{x} \in X} m(\underline{x}) \cdot p(A_3|\underline{x})$$

For fixed values of *false match rate* μ and *false non-match rate* λ, Fellegi and Sunter define the optimal linkage rule on X at levels μ and λ, denoted by $L(\mu, \lambda, X)$ as the rule for which $P(A_1|U) = \mu$, $P(A_3|M) = \lambda$, and $P(A_2|L) \leq P(A_2|L')$ for all other rules L'. Essentially, as stated in a theorem in [50], the decision rule is optimal in the sense that, for any pair of fixed upper bounds on the rates of false matches and false non-matches, the manual/clerical review region is minimized over all decision rules on the same comparison space X. By considering the likelihood ratio $l(\underline{x}) = \frac{p(\underline{x}|M)}{p(\underline{x}|U)} = \frac{m(\underline{x})}{u(\underline{x})}$, the Fellegi-Sunter linkage rule $L(\mu, \lambda, X)$ takes the form:

$$\langle \alpha, \beta \rangle \in \begin{cases} A_1 & \text{if } l(\underline{x}) > T_\mu \\ A_2 & \text{if } T_\lambda \leq l(\underline{x}) \leq T_\mu \\ A_3 & \text{if } l(\underline{x}) < T_\lambda \end{cases} \tag{4.7}$$

The cutoffs T_λ and T_μ are determined by the desired error rate bounds μ and λ on the false match rates and false non-match rates, respectively. Figure 4.2 illustrates these three regions in terms of the degree of agreement of record pair.

Under the assumption of the conditional independence of the components of the comparison vector $\underline{x}$, and by using a computationally convenient function for the likelihood ratio in (4.4) (e.g., the $\log_2$ function), the decision rule above can be written as:

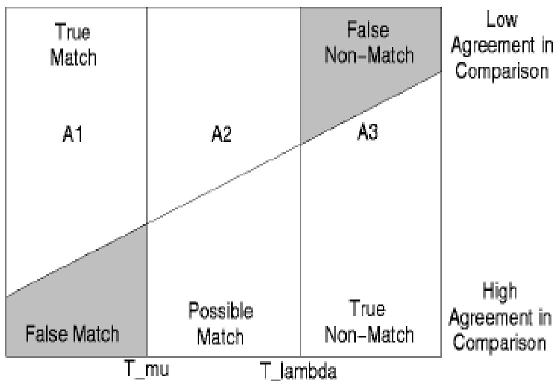


Fig. 4.2. The three regions of the Fellegi-Sunter model

$$W = W(\underline{x}) = \log_2 \left(\frac{m(\underline{x})}{u(\underline{x})} \right) = \sum_{i=1}^{n} w_i$$

where

$$w_i = \log_2 \left(\frac{m(x_i)}{u(x_i)} \right) = \log_2 \left(\frac{p(x_i|M)}{p(x_i|U)} \right)$$

W is called the *total comparison weight* associated with a generic pair $\langle \alpha, \beta \rangle$, and w_i $(i = 1, \ldots, n)$ the *individual comparison weights*.

The optimality of the decision rule (4.3), and hence the goodness of the Fellegi-Sunter probabilistic model, heavily depends on the accuracy of the estimates of the weights w_i, or more generally, of the probabilities $m(\underline{x})$ and $u(\underline{x})$. These probabilities are also called *matching parameters*.

To this purpose, Fellegi and Sunter [50] were the first to observe that the parameters needed for the decision rule in (4.3), could be obtained directly from observed data if certain simplifying assumption were made. For each $\underline{x} \in X$, they considered:

$$p(\underline{x}) = p(\underline{x}|M) \cdot p(M) + p(\underline{x}|U) \cdot p(U) \tag{4.8}$$

and noted that the proportion of pairs having representation $\underline{x} \in X$ could be computed directly from available data. If $\underline{x}$ consists of a simple agree/disagree pattern associated with three variables satisfying the conditional independence assumption that there exist vector constants (marginal probabilities) $m \equiv (m_1, m_2, \ldots, m_n)$ and $u \equiv (u_1, u_2, \ldots, u_n)$ such that, for all $\underline{x} \in X$,

$$p(\underline{x}|M) = \prod_{i=1}^{n} m_i^{x_i} (1 - m_i)^{(1-x_i)} \tag{4.9}$$

$$p(\underline{x}|U) = \prod_{i=1}^{n} u_i^{x_i}(1 - u_i)^{(1-x_i)} \tag{4.10}$$

then, Fellegi and Sunter provide the seven solutions for the seven distinct equations associated with (4.8).

However, if $\underline{x} \in X$ represents more than three variables, then it is possible to apply general equation-solving techniques such as the "*method of moments*" [76]. Because the "method of moments" has shown numerical instability in some record linkage applications [79] and with general mixture distributions [144], maximum-likelihood-based methods such as the Expectation-Maximization (EM) algorithm [44, 100] is often used.

4.4.2 Supervised Learning

The probabilistic model uses a Bayesian approach to classify record pairs into two classes M and U. This model was widely used for duplicate detection tasks, usually as an application of the Fellegi-Sunter model. While the Fellegi-Sunter approach dominated the field for more than two decades, the development of new classification techniques in the machine learning and statistics communities prompted the development of new de-duplication approaches. The supervised learning systems rely on the existence of training data in the form of record pairs, pre-labeled as matching or not.

One set of supervised learning techniques treat each record pair $\langle \alpha, \beta \rangle$ independently, similarly to the probabilistic approaches in Section 4.4.1. Cochinwala et al. in [32] describe a complete data reconciliation methodology to be used for approximate record matching. Their approach relies on the incorporation of machine learning and statistical techniques for reducing the matching complexity for large data sets. Fundamentally, this approach is based on selecting a machine learning algorithm to generate matching rules. After a specific algorithm has been selected, parameters are pruned to yield a matching rule of low complexity. Once the improved matching rule has been developed on a sample data set, it can be applied to the original data set. The authors in [32] experimented with three different learning techniques: the well-known *CART* algorithm [16], which generates classification and regression trees, a linear discriminant algorithm [72], which generates linear combination of the parameters for separating the data according to their classes, and a "vector quantization" approach, which is a generalization of nearest-neighbors algorithms. The experiments that were conducted indicate that CART has the smallest error percentage.

Bilenko et al. [12] use the *SVM-light* [80] to learn how to merge the matching results for the individual fields of the records. In this paper, the authors showed that the SVM approach usually outperforms simpler approaches, such as treating the whole record as one large field.

A typical post-processing step for the techniques so far discussed (including the probabilistic ones of Section 4.4.1) is to construct a graph for all the records in the database, linking together the matching records. Then, by using the transitivity assumption, all the records that belong to the same connected component are considered identical [104]. However, the transitivity assumption can sometimes result in inconsistent decisions. For example, $\langle \alpha, \beta \rangle$ and $\langle \alpha, \gamma \rangle$ can be considered matches, but $\langle \beta, \gamma \rangle$ not. Partitioning such "inconsistent" graphs with the goal of minimizing inconsistencies is known as an NP-complete problem [7]. Bansal et al. [7] propose a polynomial approximation algorithm that partitions such a graph, identifying automatically the cluster and the number of cluster in the data set.

Cohen and Richman [37] proposed a supervised approach in which the system learns from training data how to cluster together records that refer to the same real-world entity. The main contribution of this approach is the adaptive distance function which is learned from a given set of training examples.

Probabilistic supervised models that take into account interaction between different entity resolution decisions have been proposed for named entity recognition in natural language processing and for citation matching. McCallum and Wellner [99] employ *conditional random fields* (CRFs) for noun co-reference. Their technique is equivalent to a graph partitioning technique that tries to find the min-cut and the appropriate number of clusters for a given data set, similarly to the work [7].

The supervised clustering techniques described above, have records as nodes for the graph. Singla and Domingos [129] observed that by using attribute values as nodes, it is possible to propagate information across nodes and improve the duplicate record detection. For instance, if the records (*Google, Mountain View, CA*) and (*Google Inc., Mountain View, California*) are deemed equal, then "CA" and "California" are also equal, and this information can be useful for other record comparisons. The underlying assumption is that the only differences are due to different representations of the same entity (e.g., "Google" and "Google Inc.") and that there is no erroneous information in the attribute values (e.g., by mistake someone entering "Bismarck, ND" as the location of Google headquarter). Pasula et al. [110] propose a formal relational approach that can handle a set of transformations. This technique uses a type of Bayesian network called *Relational Probabilistic Model* [55]. While this model can handle a large number of duplicate detection problems, the use of exact inference results in a computationally intractable model. To avoid the intractability issue, the authors propose to use a *Markov Chain Monte Carlo* (MCMM) sampling algorithm.

4.4.3 Active Learning-Based Techniques

One of problems with the supervised learning techniques is the requirement for a large number of training examples. While it is easy to create a large number of training pairs that are either clearly non-duplicates or clearly duplicates, it is very difficult to generate ambiguous cases that would help to create a highly accurate classifier. Based on this observation, some duplicate detection systems used active learning techniques [38] to automatically locate such ambiguous pairs. Unlike an "ordinary" learner that is trained using a static training set, an "active" learner actively picks subsets of instances from unlabeled data, which, when labeled, will provide the highest information gain to the learner.

Sarawagi and Bhamidipaty [125] designed *ALIAS*, a learning based duplicate detection system, that uses the idea of a "reject region" (see Section 4.4.1) to significantly reduce the size of the training set. The main idea behind ALIAS is that most duplicate and non-duplicate pairs are clearly distinct. For such pairs, the system can automatically categorize them in U and M without the need of manual labeling. ALIAS requires humans to label pairs only for cases where the uncertainty is high. This is similar to the "reject region" in the Fellegi-Sunter model, which marked ambiguous cases as cases for a successive clerical review. Essentially, ALIAS works in this way. ALIAS starts with small subsets of record pairs designed for training, which have been characterized as either matched or unique. This initial set of labeled data forms the training data for a preliminary classifier. Next, the initial classifier is used for predicting the status of unlabeled pairs of records. The initial classifier will make clear determinations on some unlabeled instances but it will lack determination on most. The goal is to seek out from the unlabeled data, those instances which, when labeled, will improve the accuracy of the classifier at the fastest possible rate. Pairs whose status is difficult to determine serve to strengthen the integrity of the learner. Conversely, instances in which the learner can easily predict the status of the pairs do not have much effect on the learner. Using this technique, ALIAS can quickly learn the peculiarities of a data set and rapidly detect duplicates using only a small number of training data.

Tejada et al. [141, 142] used a similar strategy and employed decision trees to learn rules for matching records with multiple fields. Their method suggested that, by creating multiple classifier trained by using slightly different data or parameters, it is possible to detect ambiguous cases and then ask the user for feedback. The key innovation in this work is the creation of several redundant functions and the concurrent exploitation of their conflicting actions in order to discover new kinds of inconsistencies among duplicates in the data set.

4.4.4 Distance-Based Techniques

Probabilistic models require an accurate estimate of the probability parameters. As so far discussed, a possible way consists in using the training data. Indeed, for instance, when manually matched training data are available, these probability parameters can be estimated in relatively straightforward manner. However, in the absence of such training data, this estimate is not a trivial task. As a result, the probabilistic decision models can be not suitable in such situations. To this purpose, in the following we review some interesting distance-based techniques for comparing two records.

Monge and Elkan [103, 104] proposed a string matching algorithm for detecting highly similar or potentially duplicate database records. The basic idea was to apply a general purpose field matching algorithm (especially one that is able to account for gaps in the strings) to play the role of duplicate detection algorithm. Cohen [34], instead, suggested the use of the *tf-idf* weighting scheme from IR, together with the cosine similarity metric to measure the similarity of records.

Ananthakrishna et al. [5] describe a similarity metric that uses not only the textual similarity, but the "co-occurrence" similarity of two entries in a database. For instance, the entries in the state column "*CA*" and "*California*" have small textual similarity. However, the city entries "*San Francisco*", "*Los Angeles*", "*San Diego*" and so on, often have foreign keys pointing both to "*CA*" and "*California*". Therefore, it is possible to infer that "*CA*" and "*California*" are equivalent. Ananthakrishna et al. show that by using "co-occurrence" information, they can substantially improve the quality of duplicate detection in databases that use multiple tables to store the entries of a record. This approach is conceptually similar to the works [111, 41], which examine the content of fields to locate the matching fields across two tables.

Guha et al. [70] propose a distance metric that is based on ranked list merging. The main idea is that if only a field is used, the matching algorithm can easily find the best matches and rank them according to their similarity, putting the best matches first. By applying the same principle for all the fields, n ranked lists of records, one for each field, are obtained. The goal is then to create a rank of records that has minimum aggregate *rank distance* when compared to all the n lists. In [70], the authors map the problem into the *minimum cost perfect matching* problem, and develop efficient solutions for identifying the top-k matching records. The first solution is based on the *Hungarian Algorithm* [4], a graph-theoretic algorithm that solves the minimum cost perfect matching problem. Still in [70], the authors also present the *Successive Shortest Paths* algorithm that works well for smaller values of k and it is based on the idea that it is not required to examine all potential matches to identify the top-k matches.

Finally, Chaudhuri et al. [25] proposed a new framework for distance-based duplicate detection, observing that the distance thresholds for detecting real duplicate entries is different from each database tuple. In order to detect the appropriate threshold, the authors observed that entries that correspond to the same real world entity but have different representation in the database, tend to have small distances from each other (*compact set property*), and to have only a small number of other neighbors within a small distance (*sparse neighborhood property*). Furthermore, Chaudhuri et al. propose an efficient algorithm for computing the required threshold for each object in the database.

4.4.5 Equational Theory Model

Wang and Madnick [149] proposed a rule-based approach for the duplicate detection problem. For cases in which there is no global key, the authors suggest the use of rules developed by expert to derive a set of attributes that collectively serve as "key" for each record. For instance, an expert might define rules such as:

if age<18	**then** driving_license = no
	else driving_license = yes
if distance<5	**then** vehicle = bike
	else vehicle = bus

Hopefully, by using such rules are obtained unique keys that can cluster multiple records representing the same real-world entity. Lim et al. [91] also used a rule-based approach, but with the extra restriction that the result of the rules must always be correct. Therefore, the rules should not be heuristically-defined but should reflect absolute truths and serve as functional dependencies.

Hernández and Stolfo [74] further developed this idea and derived an *equational theory* that dictates the logic of domain equivalence (not simply value or string equivalence). The comparison of records to determine their equivalence is a complex inferential process that consider much more information in the compared records than the keys. For instance, if two person have similar name spellings, and these persons have same address, we might infer that they are the same person. On the other hand, let suppose that two records have exactly the same SSN (social security number), but the names and the addresses are completely different. Then, it could be either assumed that the records represent the same person who changed his name or moved, or the records represent different person and the SSN field is incorrect for one of them. Without any further information, perhaps the latter can be assumed. Obviously, the more information there is in the records, the better inference can be made. A natural approach to specify such an inference in the equational theory is the use of a declarative rule language. For instance, the following is a rule that exemplifies one axiom of the equational theory developed for an employee database:

```
for all $(r_1, r_2)$ in EMPLOYEE do
  if $r_1$.name is similar to $r_2$.name AND $r_1$.address = $r_2$.address
  then
    $r_1$ matches $r_2$;
  end if
end for
```

Note that "similar to" is measured by one of the string comparison techniques surveyed in Section 4.3, and "matches" means that those two records represent the same person.

AJAX [57] is a prototype system that provides a declarative language for specifying data cleaning programs, consisting of SQL statements enhanced with a set of primitive operations to express various cleaning transformations. AJAX provides a framework wherein the logic of a data cleaning program is modeled as a direct graph of data transformations starting from some input source data. Four types of data transformations are provided: (1) the *mapping transformations* standardizes the data, (2) the *matching transformations* finds pairs of records that probably refer to the same real-world entity, (3) the *clustering transformation* groups together matching pairs with a high similarity value, and finally (4) the *merging transformation* collapses each individual cluster into a tuple of the resulting data source.

It is worth noticing that such rule-based approaches, which require a human expert to devise matching rules, typically result in systems with high accuracy. However, this approach requires an high manual effort, by making its deployment difficult in practice. Therefore, a possible approach can consist in using a system that automatically generates matching rules from training data (supervised and active learning techniques) and then manually tune the generated rules.

4.4.6 Unsupervised Learning

As earlier discussed, the comparison space consists of comparison vectors which contain information about the differences between fields in a pair of records. Unless some information exists about which comparison vectors correspond to which category (match, non-match, and possible match), the labeling of the comparison vectors in the training data set should be done manually. One way to avoid manual labeling of the comparison vectors is to use clustering algorithms in order to group together similar comparison vectors. The key idea behind most unsupervised learning approaches for duplicate detection is that similar comparison vectors correspond to the same class.

The idea of using unsupervised approaches in the de-duplication task, has its root in the probabilistic model proposed by Fellegi and Sunter [50](see Section 4.4.1). As there discussed, when there are no training data to compute the probability estimates, it is possible to use variations of the *EM* algrithm to identify appropriate clusters in the data.

In some approaches, the use of bootstrapping techniques based on clustering to learn matching models are proposed. The basic idea, also known as *co-training* [14], is to use very few labeled data, and then use unsupervised learning techniques to label appropriately the data with unknown labels. In [147], the authors treat each entry of the comparison vector (which corresponds to the result of a field comparison) as continuous, real variable. Then, they partition the comparison space into clusters by using the *AutoClass* [27] clustering tool. AutoClass uses probability models for describing the classes, and it supports two kinds of models. The one was the multi-normal model that implements a likelihood term representing a set of real valued attributes (each with constant measurement error and without missing values) which are, given the class, conditionally independent of other attributes. The basic premise is that each cluster contains comparison vectors with similar characteristics. Therefore, all the record pairs in the cluster belong to the same class (matches, non-matches, and possible-matches). Thus, by knowing the real class of only a few vectors in each cluster, it is possible to infer the class of all vectors in the cluster, and thus mark the corresponding record pairs as matching or non-matching record pairs. Elfeky et al. [47] implemented this method in *TAILOR*, a toolbox for detecting duplicate entries in data sets (for further details see Section 4.6).

Ravikumar and Cohen [118] propose a hierarchical, graphical model in order to discover matching record pairs. The idea behind this approach is to model each field of the comparison vector as a latent binary variable which shows whether the two fields match or not. Then, the latent variable defines two probability distributions for the values of the corresponding "observed" comparison variable. Ravikumar and Cohen show that it is easier to learn the parameters of a hierarchical model than to attempt to directly model the distributions of the real-valued comparison vectors. Battacharya and Getoor [11] propose to use the *Latent Dirichlet Allocation* generative model to perform duplicate detection. In this model, the latent variable is a unique identifier for each entity in the database.

4.5 Improving the Efficiency of Duplicate Detection

In our previous discussion about methods for detecting whether two records refer to the same real entity, we focused mainly on the *quality* of the comparison techniques. Instead, in this section we are interested in the *efficiency* of the duplicate detection process.

A trivial technique for discovering matching entries in table A and B is to execute a "nested-loop" comparison, i.e., to compare every record of table A with every record in table B. Unfortunately, such a strategy requires a total of $|A| \times |B|$ comparisons. Such an approach of quadratic order (in the overall number of the input records) is prohibitively expensive even for moderately-sized tables. To this purpose, in the following we describe some techniques aiming at substantially reduce the number of required comparisons.

4.5.1 Blocking

One traditional method for identifying identical records in a database table is to scan the table and compute the value of a hash function for each record. The value of this hash function detects the "bucket" to which this record is assigned. By definition, two records that are identical will be assigned to the same bucket. Therefore, in order to locate duplicates, it suffices to compare only the records that fall into the same bucket for matches. The "classic" hashing technique cannot be used directly for approximate duplicates, since there is no guarantee that the hash value of two similar records will be the same. However, in Section 4.5.3 we study a family of hashing functions able to deal with the approximate matching problem.

An interesting counterpart of this method is named *blocking*. The term *blocking* typically refers to the procedure of subdividing files into a set of mutually exclusive subsets (blocks) under the assumption that no matches occur across different blocks. Possible approaches to achieve these blocks involve the usage of particular functions (e.g., the phonetic encodings) on highly discriminating fields (e.g., last name) and then compare only records that have similar, but not necessarily identical, fields.

Although blocking can increase the speed of the comparison process, it can lead to an increased number of false mismatches due to the failure of comparing records that not agree on the blocking field. Moreover, it can also lead to an increased number of missed matches due to possible errors in the blocking step that placed entries in wrong buckets.

Phonetic Encoding

Character-level and token-based similarity metrics, as viewed in Section 4.3, focus on the string-based representation of the database records. However, strings may be phonetically similar even if they are not similar in a character or token level. Therefore, the idea behind the phonetic encoding is that words phonetically similar share the same encode.

The most common coding scheme is known as the *Russell Soundex* code[1]. Soundex is based on the assignment of identical code digits to phonetically similar groups of consonants and is used mainly to match surnames. The rules of Soundex code are as follows:

1. The first letter of the surname is not coded and serves as the prefix letter. W and H are ignored completely.
2. Other letters are coded as follows:

   - $B, F, P, V \rightarrow 1$.
   - $C, G, J, K, Q, S, X, Z \rightarrow 2$.
   - $D, T \rightarrow 3$.
   - $L \rightarrow 4$.
   - $M, N \rightarrow 5$.
   - $R \rightarrow 6$.

3. A, E, I, O, U and Y are not coded but serve as separators (see below).

[1] U.S. patents 1,261,167 and 1,435,663. Available at http://patft.uspto.gov/netahtml/srchnum.htm.

4. The sequences of identical codes are consolidated by keeping only the first occurrence of the code.
5. Separators are dropped.
6. The letter prefix and the three first codes are kept. It there are fewer than three codes, zeros are added.

The code is designed primarily for Caucasian surnames, but works well for names of many different origins. However, when the names are principally of east-asian origin, this code deteriorates because much of the discriminating power of these names resides in the vowel sounds, which the code ignores.

New York State Identification and Intelligent System (NYSIIS), proposed by Taft [139], differs from Soundex in that it retains information about the position of vowels in the encoded word, by converting most vowels to the letter *A*. Furthermore, NYSIIS does not use numbers to replace letters; instead it replaces consonants with other phonetically similar letters, thus returning a purely alphabetic code (no numeric component). Taft compared Soundex with NYSIIS, using a name database of New York State and concluded that NYSIIS is 98.72% accurate, while Soundex is 95.99% accurate for locate surnames.

Oxford Name Compression Algorithm (ONCA) [60] is a two stage technique, designed to overcome most of unsatisfactory features of pure Soundex algorithm. In the first step, ONCA uses a British version of the NYSIIS method of compression. Then, in second step, the transformed and partially compressed name is "Soundexed" in the usual way. This two-stage technique has been used successfully for blocking similar names together.

Philips [112] proposed the *Metaphone* algorithm as a better alternative to Soundex. Philips suggested using 16 consonant sounds that can describe a large number of sounds used in many English and non-English words.

Double Metaphone [113] is a better version of Metaphone, improving some encodings choices made in the initial Metaphone and allowing multiple encodings for names that have several pronunciations. For such cases, all possible encodings are tested when trying to retrieve similar names. The introduction of multiple phonetic encodings greatly enhances the matching performances, with rather small overhead.

4.5.2 Sorted Neighborhood Approach

In [73] the authors describe the so-called *sorted neighborhood* approach. The idea behind this approach consists in sorting the entire data-set in order to bring all duplicate records close together. Indeed, after the sort, the comparison of records can be restricted to a small neighborhood within the sorted list. The effectiveness of this approach is strictly related to the quality of the chosen key used in the sorting. Choosing a few discriminant key will cause that there is a very small possibility that a record will end up close to a matching records after the sorting. Essentially, the sorted neighborhood method can be summarized in three phases:

1. **Create key** A key for each record in the list is computed by extracting relevant fields or portions of fields.
2. **Sort data** The records in the database are sorted by using the key found in the first step. A sorting key is defined as a sequences of attributes, or a sequence of substrings within the attributes, appropriately chosen from the record. Attributes that appear first in the key have higher priority than those that appear subsequently.
3. **Merge** A fixed size window is moved through the sequential list of records in order to limit the comparisons for matching records to those records in the window. If the size of the window is w records, then every new record entering the window is compared with the previous $w - 1$ records to find "matching" records. The first record in the window slides out of it.

The sorted-neighborhood approach can be viewed as an extension, able to identify approximate duplicates, of the standard method of detecting exact duplicates in a database table described in [13]. Here, the idea is simply to sort the table and then to check if neighboring tuples are identical.

Sorting and then matching within a window is the essential approach of a *Sort Merge Band Join* as described in [45]. In this paper, the sort and merge phase can be combined in one step. The main difference between [73] and [45] resides in the use of a complex domain-dependent function (the *equational theory*, see Section 4.4.5) to determine if records under consideration match.

Although, sorting the data may not be the dominant cost of the sorted neighborhood approach, the authors, still in [73], consider an alternative technique, called *clustering method*, for sorting the entire databases. They propose to partition the data-set into independent clusters using a key extracted from the data. Then, for each one of the cluster generated in that way, the sorted neighborhood method can be applied.

In order to improve the accuracy of the sorted neighborhood method, the authors in [74] describe a *multi-pass* approach. The multi-pass strategy is based on the execution of several independent runs of the sorted neighborhood method, each time using a different key and a relatively small window. Each independent run will produce a set of pairs of records that can be merged. The transitive closure is then applied to those pairs of records. Therefore, the final result will be a union of all pairs discovered by all independent runs (with no duplicates), plus all those pairs inferred by means of the application of the transitive closure to the records matched in the different passes.

4.5.3 Locality Sensitive Hashing Functions

The *Locality Sensitive Hashing* (LSH) functions were introduced by Indyk and Motwani [77] for efficiently answering approximate nearest neighbor queries on a collection of objects.

By definition, a *locality sensitive hashing* function H for a generic set S equipped with a distance function D, is a function which bounds the probability of collisions to the distance between elements. Formally, given H, for each pair $\langle\alpha,\beta\rangle \in S$ and value ϵ, there exists values P_1^ϵ and P_2^ϵ such that:

- if $D(\alpha,\beta) \leq \epsilon$ then $Pr[H(\alpha) = H(\beta)] \geq P_1^\epsilon$
- if $D(\alpha,\beta) > \epsilon$ then $Pr[H(\alpha) = H(\beta)] > P_2^\epsilon$

The key idea behind this approach that make it helpful in the deduplication scenario, is that two similar objects (according to a similarity metric) "processed" with the same locality sensitive function H, probably tend to have the same hashing encode. Intuitively, the LSH functions can be exploited to build hash-based indexing schemes able to put similar records into the same bucket.

The *min-wise independent permutations* [19] represent a particular instance of the LSH function family. The theory of min-wise independent permutations became popular since it is essential to the algorithm used by AltaVista web index software to detect and filter near duplicate documents [20].

Formally, an *exactly min-wise independent permutation* is a coding function π of a set X of generic items, such that, for each $x \in X$:

$$Pr[min(\pi(X)) = \pi(x)] = \frac{1}{X} \tag{4.11}$$

In other words, is required that all items of any fixed set X have equal chance to become the minimum element of the image of X under π. A randomly chosen min-wise independent permutation π, for each two set X and Y, ensure that:

$$Pr[min(\pi(X)) = min(\pi(Y))] = \frac{|X \cap Y|}{|X \cup Y|} \tag{4.12}$$

On the basis of this property, it becomes clear that two records α and β sharing much fields (i.e., with a high Jaccard similarity), in a probabilistic manner tend to have the same hashing encode and then to fall into the same bucket.

In order to implement such a bucketing scheme, the key point is the definition of a proper family of min-wise independent permutations. However, in practice we have to deal with the sad consideration that is hard to define a family of exactly min-wise independent permutations. Thus, we are led to allow certain relaxations and to consider smaller families of permutations that still satisfy the min-wise independence condition given by Equation (4.11), since min-wise independence is necessary and sufficient for Equation (4.12) to hold. Essentially, small relative errors can be accepted, and a family of *approximatively min-wise independent permutations* with relative error δ can be defined as:

$$\left| Pr[min(\pi(X)) = \pi(x)] - \frac{1}{X} \right| \leq \frac{\delta}{X} \tag{4.13}$$

In other words is required that all the elements $x \in X$ have only almost equal chance to become the minimum element of the image of X under π.

Subsequently, in the Chapter 6, we exploit a family of "practically" min-wise independent permutations (already used by Broder et al. in [20]) in order to build an effective hashing scheme able to cluster duplicate records in a database.

4.5.4 Clustering and Canopies

Priority Queue Algorithm

The authors in [104] try to improve the performance of a basic "nested-loop" record comparison, by assuming that duplicate detection is transitive. This means that if the record α is deemed duplicate of the record β and β is deemed duplicate of the record γ, then α and γ are also duplicates. Under the assumption of transitivity, the problem of detecting duplicates in a database can be described in terms of determining the connected components of an undirected graph. At any time, the connected components of this graph correspond to the transitive closure of the "is a duplicate of" relationships discovered so far. The main idea behind this approach consists in querying the graph before to apply any expensive pairwise record matching algorithm to two records. Indeed, if both records are in the same connected component, then it has been previously determined that they are duplicate, and the comparison is not needed. If they belong to different components, then it is unknown whether they match or not. If comparing the two records results in a match, a new edge between the vertices that corresponds to the records compared, is inserted.

There is a well-known data structure that efficiently solves the problem of determining and maintaining the connected components of an undirect graph: the *union-find* data structure [140, 39]. This data structure provides two operations:

- $Union(x, y)$ combines the sets that contain node x and node y (S_x and S_y respectively) into a new set that is their union $S_x \cup S_y$. A representative for the union is chosen and the new set replaces S_x and S_y in the collection of disjoint sets.
- $Find(x)$ returns the representative of the unique set containing x.

The proposed algorithm uses two passes. The first one, treats each record as one long string and sorts all the records lexicographically, reading from left to right. The second pass does the same reading from right to left. The algorithm uses a *priority queue* of sets of records belonging to the last few clusters detected. The algorithm scans the database sequentially and determines whether each scanned record is or is not a member of a cluster represented in the priority queue. To determine cluster membership, the algorithm uses the $Find$ operation above described. If the record is already a member of any cluster kept in the priority queue, then the next record is scanned. If the record is not a member of any cluster, then the record is compared with the representative of each cluster in the priority queue using the *Smith-Waterman* matching algorithm (see Section 4.3.1). If one of this comparisons succeeds, then the record belongs to this cluster and the *Union* operation is performed on the two sets. On the other hand, if all comparisons fail, then the record must be a member of a new cluster not currently represented in the priority queue. Essentially, the concept behind this approach is that if a record α is not similar to a record β already in the cluster, then it will not match the other members of the cluster either. Therefore, it is clear that the total number of record comparisons will be reduced.

Canopies

In [98] the authors propose a new efficient clustering technique for speeding up the duplicate detection process. The key idea of the *canopy algorithm* is that one can greatly reduce the number of distance computations required for clustering, by first cheaply partitioning the data into overlapping subsets, and then only measuring distances among pairs of records that belong to a common subset.

More in detail, this approach works in two stages. In the first stage, a cheap distance measure that divides the data into overlapping subsets called "*canopies*" is used. A canopy is simply a subset of records that, according to the approximate similarity measure, are within some distance threshold from a "central point". Significantly, a record may appear under more than one canopy, and every element must appear in at least one canopy. This is in contrast with the blocking strategy that requires hard, non-overlapping partitions. In the second stage, a traditional clustering algorithm (e.g., Greedy agglomerative clustering or K-means) using a more expensive similarity metric is executed. However, significant computation is saved by eliminating all of distance comparisons among records that do not fall within a common canopy.

Essentially, the assumption behind the use of this method is that there is an inexpensive similarity function that can compensate for another, more expensive function. Therefore, very important is the choice of this cheap distance metric. In some cases, this choice could be trivial. For example, if the data consists of a large number of hospital patient records including diagnoses and payment histories, a cheap measure of similarity between the patients might be "1" if they have a diagnosis in common and "0" if they do not. In this case canopy creation is very easy: people with a common diagnosis fall in the same canopy.

In the following, a brief review of some well-known approaches in literature, regarding the choice of an approximate distance to build canopies, will be described. In [37] the authors propose the *tf-idf* similarity metric as a canopy distance, and then use multiple (expensive) similarity metrics to infer whether two records are duplicates. Gravano et al. in [65] propose to use the string lengths and the number of common *q-grams* of two strings as canopies for the edit distance metric, which is expensive to compute in a relational database. The advantage of this technique is that the canopy functions can be evaluated efficiently using vanilla SQL statements. In a similar fashion, Chaudhuri et al. [24] propose using an *indexable* canopy function for easily identifying similar tuples in a database. Baxter et al. [9] perform an experimental comparison of canopy-based approaches with traditional blocking and show that the flexible nature of canopies can significantly improve the quality and speed of duplicate detection process.

4.5.5 Set Joins

Another direction towards efficiently implementing data de-duplication operations is to speed-up the execution of set operations. Large number of similarity metrics, discussed in Section 4.3, use set operations as part of the overall computation. Running set operations on all pair combinations is a computationally expensive operation and is typically unnecessary. For de-duplication applications, the interesting pairs are only those in which the similarity value is high. Many techniques use this property and suggest algorithms for fast computation of set-based operations on a set of records.

Cohen [34] proposed using a set of in-memory inverted indexes with a search algorithm to locate the top-k most similar pairs, according to the cosine similarity metric. Soffer et al. [135], mainly in the context of information retrieval, suggest pruning the inverted index, removing terms with low weights since they do not contribute much to the computation of the *tf-idf* cosine similarity. Gravano et al. [66] present an SQL-based approach that is analogous to the approach in [135], and allows fast computation of cosine similarity within an RDBMS. Mamoulis [94] presents techniques for efficiently processing a set join in a database, focusing on the containment and non-zero-overlap operators. Mamoulis shows that inverted indexes are tipically superior to the approaches based on signature files, confirming earlier comparison studies [160]. The authors in [126] extend the set joins approach to a large number of similarity predicates that use set joins. The *Probe-Cluster* approach here proposed, works well in environments with limited main memory, and can be used to compute efficiently a large number of similarity predicates, in contrast to the previous approaches which were tuned for a small number of similarity predicates (e.g., set containment or cosine similarity). Furthermore, *Probe-Cluster* returns exact values for the similarity metrics, in contrast to previous approaches which use approximation techniques.

4.6 Systems

Recently, several industrial tools for de-duplicating data was developed. In this section we review some of such tools.

The *Febrl* system[2] (Freely Extensible Biomedical Record Linkage) is an open-source data cleaning toolkit, and it has two main components: the first component deals with data standardization and the second perform the actual duplicate detection. The data standardization relies mainly on *hidden Markov models* (HMMs). Therefore this system typically requires a training phase to correctly parse the database entries. For the actual duplicate detection phase, Febrl implements a variety of string similarity metrics: Jaro, Edit distance, and q-gram distance (see Section 4.3.1). Moreover, Febrl supports phonetic encodings (Soundex, NYSIIS, and Double Metaphone reviewed in Section 4.5.1) to detect similar names. Since phonetic similarity is sensitive to errors in the first letter of a name, Febrl also computes phonetic similarity using the reversed version of the name string, then avoiding the "first-letter" sensitivity problem.

TAILOR [47] is a felxible record matching toolbox, which allows the users to apply different duplicate detection methods on the data sets. The flexibility of using multiple models is useful when the users do not know which duplicate detection model will perform most effectively on their particular data. TAILOR follows a layered design, separating comparison functions from the duplicate detection logic. Furthermore, the execution strategies, which aim is to improve the efficiency, are implemented in a separate layer, making the system more extensible than systems that rely on monolithic designs. Finally, TAILOR report statistics such as estimated accuracy and completeness, which can help the users understand how a duplicate detection model performs over a data set.

WHIRL[3] is a duplicate detection system freely available for academic and research purposes. WHIRL uses the *tf-idf* token-based similarity metric to identify similar strings within two lists. The *Flamingo* project[4] is a similar tool that takes as input two string lists and returns the strings pair that are within a fixed edit distance threshold.

WizSame by WizSoft is a product (not freely available) that allows the discovery of duplicate records in a database. The matching algorithm used in this tool is very similar to $SoftTF - IDF$ (see Section 4.3.2): two records match if they contain a significant fraction of identical or similar words, where similar are the words that are within an edit distance of 1.

BigMatch [155] is the duplicate detection program used by the U.S. Census Bureau. It relies on several blocking strategies to identify potential matches between the records of two relations, and scales well for very large data sets. The only requirements is that one of the two relations should fit in memory. This is possible even for relations with 100 million records. More specifically, the purpose of the BigMatch tool is to function as a preprocessor,that is rather than performing sophisticated duplicate detection, it generates a set of

[2] available at http://sourceforge.net/projects/febrl
[3] available at http://www.cs.cmu.edu/˜wcohen/whirl/
[4] available at http://www.ics.uci.edu/˜flamingo/

candidates pairs that should be then processed by more appropriate record linkage algorithms.

Finally, to the best of our knowledge, no database vendors provide sufficient tools for duplicate record detection task. Until now, most of the efforts are focused on creating easy-to-use *ETL* (Extraction, Transformation, Loading) tools, that can standardize database records and fix min errors, mainly in the context of address data. Today, another typical function of the tools is the ability to use reference tables and standardize entities having multiple well-known representations. For instance, the computer science journal "TODS" is also frequently referred as "ACM TODS" or as "Transactions on Database Systems". A very recent positive step is the insertion of multiple data cleaning operators within *Microsoft SQL Server Integration Services*, which is part of Microsoft SQL Server 2005. For instance, SQL Server 2005 consents to perform "fuzzy matches" and implements "error-tolerable indexes" allowing for fast execution of such approximate lookups. In order to discover possible matching records, the $SoftTF - IDF$ similarity metric is adopted.

4.7 Conclusions

In this chapter, we presented a survey of the existing techniques used for detecting non-identical duplicate entries in database records. As database systems are becoming more and more pervasive, data de-duplication is going to be the cornerstone for managing systems which accumulate vast amounts of errors on daily basis. However, several issues are still open in the de-duplication scenario.

First of all, it is unclear which metrics and techniques should be used for a given duplicate detection task. Some studies exist in literature that compare the effectiveness of the various distance metrics so far presented. For instance, Yancey [156] shows that the *Jaro-Winkler* metric works well in name matching tasks for data coming from U.S. Census. Bilenko et al. [12], by comparing the effectiveness of character-based and token-based similarity metrics, showed that the $SoftTF - IDF$ metric works better than any other metric. Actually, no single metric is suitable for all data sets. Metrics that demonstrate robustness and high-performances when deal with some data sets, can perform poorly on others. Therefore, the duplicate record detection can be considered as a highly data-dependent task. Under this perspective, the problem of choosing the best method for duplicate record detection is very similar to the problem of *model selection* and *performance prediction* for Data Mining.

Currently, two main research areas study the duplicate detection problem. The research on databases tends towards relatively simple and fast "ad-hoc" duplicate detection techniques able to deal with millions of records. Such approaches typically rely on the existence of training data, and emphasize efficiency over effectiveness. On the other hand, research in machine learning and statistics aims to develop more sophisticated matching techniques relying on probabilistic models. However, probabilistic inference techniques are practical today only for data sets that are one or two orders of magnitude smaller than the data sets handled by ad-hoc techniques.

A promising direction for the future research could consist in devising techniques that combine the best of both worlds.

5

An Incremental Clustering Approach to Data Reconciliation

5.1 Introduction

Recognizing similarities in large collections of data is a major issue in the context of information integration systems. An important challenge in such a setting is to discover and properly manage duplicate tuples, i.e., syntactically different tuples which are actually identical from a semantical viewpoint, for they referring to the same real-world entity. There are several application scenarios involving this important task. A typical example consists in the reconciliation of demographic data sources in a data warehousing setting. Names and addresses can be stored in rather different formats, thus raising the need for an effective reconciliation strategy which could be crucial for effective decision making. In such cases the problem is the analysis of a (typically large) volume of small strings, in order to reconstruct the semantic information on the basis of the few syntactic information available. Let consider, e.g., a banking scenario, in which the main interest is to rank the credit risk of a customer by looking at the past insolvency history. Since information about payments may come from different sources, each of which using a possibly different format for storing the data, de-duplicating demographic tuples is crucial in order to correctly monitor customer behavior.

In such application scenarios, there is in general little syntactic information associated with each tuple. A tuple is usually obtained from a legacy system, and is represented by a (small) sequence of strings, and no typing information is available. Thus, by assuming that each possible string represents a dimension along which the information contained in a tuple can projected, tuples represent a small informative content in a high-dimensional space.

In the literature, the problem of tuple de-duplication has been dealt with mainly from an accuracy viewpoint, by taking care to the minimization of incorrect matchings (for a detailed analysis of the current literature, see Chapter 4). However, efficiency and scalability issues do play a predominant role in many application contexts where large data volumes are involved, especially when the object-identification task is part of an interactive application, calling for short response times.

Consider again the banking scenario: data collected on a daily basis typically consists of 500,000 instances, representing credit transactions performed by customers throughout the various agencies. In such a case the simple solution of comparing all database instances in a pairwise way, according to some given similarity measure, is clearly impractical. For example, for a set of 30,000,000 tuples (i.e., data collected in a 2 months-monitoring), this simple method would require $O(10^{15})$ (a quadrillion) comparisons. This is clearly infeasible.

In general, the large volume of involved data imposes severe restrictions on the design of data structures and algorithms for data de-duplication, and disqualifies any approach requiring quadratic time in the database size or producing many random disk accesses and continuous paging activities. Thus, in this chapter our main objective is to devise a scalable method for duplicate detection that can be profitably applied to large databases. We approach the problem from a *clustering* perspective: given a set of tuples, recognizing subsets (clusters) of tuples such that intra-cluster similarity is high, and inter-cluster similarity is low. Three main features make the problem at hand significantly different from traditional approaches:

- tuples are represented as (small) sequences of tokens, where the set of possible tokens is high;
- the number of clusters is too high to allow the adoption of traditional clustering techniques, and
- the streaming (constantly increasing) nature of the data imposes linear-time algorithms for clustering.

The solution we propose essentially relies on an efficient clustering technique that allows to discover all clusters containing duplicate tuples in an incremental way. The core of the approach is the usage of a suitable indexing technique which, for any newly arrived tuple μ, allows to efficiently retrieve a set of tuples in the database which are likely mostly similar μ, and hence are expected to refer to the same real-world entity associated with μ. The proposed indexing technique is based on a hashing scheme, which tends to assign objects with high similarity to the same buckets.

More in detail, in this chapter we propose an indexing scheme tailored to a set-based distance function, and the management of each tuple was faced at a coarser granularity. In Chapter 6, we next extend and improve this proposal in both effectiveness and efficiency, by allowing for a direct control on the degree of granularity needed to properly discover the actual neighbors (duplicates) of a tuple.

This chapter is organized as follows. In Section 5.2, we first formalize the discovery of duplicate objects as a specific clustering problem. Then, Section 5.3 illustrates an efficient technique that is able to discover all clusters containing duplicate tuples in an incremental way. The core of this approach is the effective hash-based indexing technique proposed in Section 5.4. In Section 5.5, we show experimental results demonstrating that the hash-based method allows to obtain considerable improvement in efficiency w.r.t. a state-of-art indexing approach (M-tree). In Section 5.6, we shortly perform a qualitative comparison with some relevant proposals in literature mostly related to our approach. Finally, in Section 5.7 some conclusions are drawn.

5.2 Problem statement

In the following we introduce the basic notation and some preliminary definitions. An item domain $\mathcal{M} = \{a_1, a_2, \ldots, a_m\}$ is a collection of items. We assume m to be very large: typically, $\mathcal{M}$ represents the set of all possible strings available from a given alphabet. Moreover, we assume that $\mathcal{M}$ is equipped with a distance function $dist_{\mathcal{M}}(\cdot, \cdot) : \mathcal{M} \times \mathcal{M} \mapsto [0, 1]$, expressing the degree of dissimilarity between two generic items a_i and a_j.

A tuple μ is a subset of $\mathcal{M}$. An example tuple is:

{`Alfred, Whilem, Salisbury, Hill, 3001, London`}

representing registry information about a subject. Notice that, a more appropriate representation can take into account a relational schema in which each tuple fits. For example, in the above schema, a more informative setting requires to segment the tuples into the fields `NAME`, `ADDRESS`, `CITY`, and to associate an itemset to each field: μ[`NAME`] = {`Alfred`, `Whilem`}, μ[`ADDRESS`] = {`Salisbury, Hill, 3001`}, μ[`CITY`] = {`London`}.

For ease of presentation, we shall omit such details: the results which follow can be easily generalized to such a similar context.

We assume that the set of all tuples is equipped with a distance function, $dist(\mu, \nu) \in [0, 1]$, which can be defined for comparing any two tuples μ and ν, by suitably combining the distance values computed through $dist_{\mathcal{M}}$ on the values of matching fields. In the following, we assume that both $dist(\mu, \nu)$ and $dist_{\mathcal{M}}$ are defined in terms of the *Jaccard* coefficient.

The core of de-duplication problem can be roughly stated also in this form: detecting, within a database $\mathcal{DB} = \{\mu_1, \ldots, \mu_N\}$ of tuples, a suitable partitioning $\{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ of the tuples, such that for each group $\mathcal{C}_i$, intra-group similarity is high and extra-group similarity is low. For example, the dataset:

μ_1	`Jeff, Lynch, Maverick, Road, 181, Woodstock`
μ_2	`Anne, Talung, 307, East, 53rd, Street, NYC`
μ_3	`Jeff, Alf., Lynch, Maverick, Rd, Woodstock, NY`
μ_4	`Anne, Talug, 53rd, Street, NYC`
μ_5	`Mary, Anne, Talung, 307, East, 53rd, Street, NYC`

can be partitioned into $\mathcal{C}_1 = \{\mu_1, \mu_3\}$ and $\mathcal{C}_2 = \{\mu_2, \mu_4, \mu_5\}$.

This is essentially a clustering problem, but it is formulated in a specific situation, where there are several pairs of tuples in $\mathcal{DB}$ that are quite dissimilar from each other. This can be formalized by assuming that the size of the set $\{\langle\mu_i, \mu_j\rangle \mid dist(\mu_i, \mu_j) \simeq 1\ \}$ is $O(N^2)$: thus, we can expect the number K of clusters to be very high – typically, $O(N)$. Moreover, we intend to cope with the clustering problem in an incremental setting, where a new database $\mathcal{DB}_\Delta$ must be integrated with a previously reconciled one $\mathcal{DB}$. Practically speaking, the cost of clustering tuples in $\mathcal{DB}_\Delta$ must be (almost) independent of the size N of $\mathcal{DB}$.

5.3 A Clustering Approach to Data Reconciliation

The key issue in the problem described above, is the capability of detecting cluster membership for a generic tuple μ_i by means of a minimal number of comparisons. This can be achieved by exploiting a proper k-Neirest Neighbor technique, in which the k nearest neighbors of μ_i are efficiently extracted from the given database. Algorithm 5.1 summarizes our solution to the data reconciliation problem. Notably, the clustering method is parametric w.r.t. the distance function used to compare any two tuples, and is defined in an incremental way, for it allowing to integrate a new set of tuples into a previously computed partition. In fact, the algorithm receives a database $\mathcal{DB}$ and an associated partition $\mathcal{P}$, besides the set of new tuples $\mathcal{DB}_\Delta$; as a result, it will produce a new partition $\mathcal{P}'$ of $\mathcal{DB} \cup \mathcal{DB}_\Delta$, obtained by adapting $\mathcal{P}$ with the tuples from $\mathcal{DB}_\Delta$. To this purpose, each tuple in $\mathcal{DB}_\Delta$ is associated with a cluster in $\mathcal{P}$, detected through a sort of *nearest-neighbor* classification scheme. The basic intuition in the proposed approach is that, since the number of clusters is high (typically $O(N)$), then it suffices to compare few "close" neighbors in order to obtain the appropriate cluster membership.

In more detail, for each tuple μ_i in $\mathcal{DB}$ to be clustered, the neighbors of μ_i are retrieved by means of procedure kNearestNeighbor, which performs a search for the k most prominent neighbors and using μ_i as query object. The cluster membership for μ_i is determined by calling the MostLikelyClass procedure, which estimate the *most likely* cluster among the ones associated with the neighbors of μ_i. Such an estimation is carried out via a *voting* strategy, where each neighbor μ_j of μ_i votes for the cluster it belongs to, by adding a contribution $\frac{1}{dist(\mu_i,\mu_j)}$ to the score of its cluster.

The score of each cluster is normalized by dividing it by the number of tuples that voted for the cluster; tuple μ_i is then assigned to the cluster receiving the highest normalized score, provided that this is greater than a given threshold (in our usual setting we use 0.5 as threshold). If such a cluster does not exist, μ_i is estimated not to belong to any of the existing clusters with a sufficient degree of certainty, and hence it is assigned to a newly generated cluster. Finally, procedure Propagate is meant to scan the neighbors of μ_i in order

```
Generate-Clusters($\mathcal{P}$,$\mathcal{DB}_\Delta$,$k$)
  Output: A partition $\mathcal{P}'$ of $\mathcal{DB} \cup \mathcal{DB}_\Delta$;
  1:  $\mathcal{P}' \leftarrow \mathcal{P}$; $\mathcal{DB}' \leftarrow \mathcal{DB}$;
  2:  Let $\mathcal{P}' = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ and $\mathcal{DB}_\Delta = \{\mu_1, \ldots, \mu_n\}$;
  3:  for $i = 1 \ldots n$ do
  4:    $neighbors \leftarrow$ kNearestNeighbor($\mathcal{DB}', \mu_i, k$);
  5:    $\mathcal{C}_j \leftarrow$ MostLikelyClass($neighbors, \mathcal{P}'$);
  6:    $\mathcal{DB}' \leftarrow \mathcal{DB}' \cup \{\mu_i\}$;
  7:    if $\mathcal{C}_j = \emptyset$ then
  8:      create a new cluster $\mathcal{C}_{m+1} = \{\mu_i\}$;
  9:      $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{\mathcal{C}_{m+1}\}$;
  10:   else
  11:     $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{\mu_i\}$;
  12:     Propagate($neighbors, \mathcal{P}'$);
  13:   end if
  14: end for
Propagate($S$,$\mathcal{P}$)
  P1: for all $\mu \in S$ do
  P2:   $neighbors \leftarrow$ kNearestNeighbor($\mathcal{DB}, \mu, k$);
  P3:   $\mathcal{C} \leftarrow$ MostLikelyClass($neighbors, \mathcal{P}$);
  P4:   if $\mu \notin \mathcal{C}$ then
  P5:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mu\}$;
  P6:     Propagate($neighbors, \mathcal{P}$);
  P7:   end if
  P8: end for
```

Fig. 5.1. Clustering algorithm

to possibly revise their cluster memberships, since in principle they could be affected by the insertion of μ_i. In particular, for each tuple μ_j in its input set, the membership of μ_j is estimated again by MostLikelyClass, and, if it does not agree with the cluster actually containing μ_j, the membership of μ_j is updated, and Propagate is recursively applied to the neighbors of μ_j. In principle, this task might be iterated over each reassigned tuple, and could then be of linear complexity w.r.t. the size of $\mathcal{DB}$. Notice that, in typical Entity Resolution settings, where clusters are quite distant from each other, the propagation affects only a reduced number of tuples, and ends in a low number of iterations.

It is worth remarking that the complexity of Algorithm 5.1, given the size N of $\mathcal{DB}$ and M of $\mathcal{DB}_\Delta$, depends on the three major tasks: the search for neighbors (line 4, having cost n), the voting procedure (line 5, with a cost proportional to k), and the propagation of cluster labels (line 12, having a cost proportional to n, based on the discussion above). As they are performed for each tuple in $\mathcal{DB}_\Delta$, the overall complexity is $O(M(n + k))$. Since k is $O(1)$, it follows that the main contribution to the complexity of the clustering procedure is due to the cost $O(n)$ of the kNearestNeighbor procedure. Therefore, the main efforts towards computational savings are to be addressed in designing an efficient method for neighbor search. Our main goal is doing this task by minimizing the number of accesses to the database, and avoiding the computation of all pair-wise distances.

5.4 Optimizing the Search for Neighbors

As above described, the procedure kNearestNeighbor plays a fundamental role in the performance of the algorithm specified in Figure 5.1. Therefore, in order to efficiently retrieve the neighbors of the current tuple, we need to resort to an indexing scheme that can support the execution of similarity queries, and can be incrementally populated with new tuples. In the following we describe two main indexing schemes which have been investigated for being embedded in our approach: the *M-tree index*, specifically designed for searching in metric spaces, and a novel index based on hashing. Both these structures support an efficient implementation on secondary memory, in order to ensure the scalability of the overall approach even in presence of large data volumes.

5.4.1 Exploiting an M-Tree

An *M-tree* [30] is an original index/storage structure, which looks like a n-ary tree, and exhibits the following major features:

- it is a paged, balanced, and dynamic secondary-memory structure able to index data sets from generic metric spaces.
- similarity range and nearest-neighbor queries can be performed and results can be ranked with respect to a given query object.
- query execution is optimized to reduce both the number of pages read and the number of distance computations.

The tree structure is capable of indexing generic objects, provided that a suitable distance metric is defined for comparing them. The interesting point of the M-tree structure is that it represents a balanced hierarchical clustering structure, in which each cluster has a fixed size (related to the size of a page to be stored on disk). Thus, a similarity search can be accomplished by traversing the tree, and ignoring subtrees reputed uninteresting for the search purpose. Indeed, each entry in a non-leaf node stores a pointer to a node t at the next lower level along with summary information about the contents of the subtree rooted in t. In particular, entries store *routing objects*, i.e., database objects representing that are given a routing role by some specific promotion algorithm. Each routing object O_r is associated with a pointer referencing the root of a sub-tree $T(O_r)$, named the *covering tree* of O_r, and with a *covering radius* $r(O_r)$, guaranteeing that all objects in the covering tree of O_r are within the distance $r(O_r)$ from O_r.

A similarity search with a given object query q and a search-radius ϵ, can be efficiently performed on the M-tree with a straightforward traversal. Indeed, at each non-leaf node storing a routing object O_r, the comparison between $distance(q, O_r)$ 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). It is worth noticing that the values actually stored for covering radii, hence, strongly affect the performances of the index: the smaller the radii, the more selective and effective will be a search.

The above described peculiarities, combined with its generality, make the M-tree particularly worthwhile to consider in our setting. However, the benefits of this indexing structure is likely to degrade in a typical entity resolution scenario. Notably, in such a setting, most of the internal nodes in the tree tend to correspond to a quite "heterogeneous" set of tuples, and hence a high number of levels, i.e., nearly linear in the number of distinct entities, is required to suitably partition the whole dataset. In other words, the presence of several heterogenous objects raises the size of the covering radii, especially at higher levels. This causes a general degradation in the performance of the tree structure.

5.4.2 Hash-based Indexing Schema

A substantial improvement to the performance of the GENERATE-CLUSTERS algorithm can be achieved by exploiting a hash-based indexing scheme, which could guarantee the execution of neighbor searches in a time that does not depend on the number of database tuples, and can be incrementally updated with new tuples.

The basic idea is to map any tuple to a proper set of features, so that the similarity between two tuples can be evaluated by simply looking at their respective features. Under this perspective, the role of the hashing schema is to maintain the association between tuples and the corresponding features, so that the neighbors of a tuple μ can be efficiently computed, by simply retrieving the tuples that appear in the same buckets which correspond to μ. To this purpose, a hash-based index structure, simply called *Hash Index*, was introduced, which consists of a pair $H = \langle FI, ES \rangle$, where:

- ES, referred to as *External Store*, is a storage structure devoted to manage a set of tuple buckets through an optimized usage of disk pages: each bucket gathers tuples that are estimated to be similar to each other, for they sharing a relevant set of properly defined features;
- FI, referred to as *Feature Index*, is an indexing structure which, for each given feature s, allows to efficiently recognize all the buckets in ES that contain tuples exhibiting s.

Figure 5.2 illustrates how such an index can be exploited for performing nearest-neighbor searches, and then supporting the whole clustering approach previously described. The algorithm works according to the number k of desired neighbors.

```
kNearestNeighbors($\mathcal{DB}$,$\mu$,$k$)
 1: Let $S = \{s \mid s$ is a relevant feature of $\mu\}$;
 2: $\mathcal{N} \leftarrow \emptyset$;
 3: while $S \neq \emptyset$ do
 4:    $x = S.Extract()$;
 5:    $h \leftarrow FI.Search(x)$;
 6:    if $(h = 0)$ then
 7:       $h \leftarrow FI.Insert(x)$;
 8:    else
 9:       while $\nu = ES.Read(h)$ do
10:          if $\mathcal{N}.size < k$ or $dist(\mu,\nu) < \mathcal{N}.MaxDist()$ then
11:             $\mathcal{N}.Insert(\nu, dist(\mu,\nu))$;
12:          end if
13:       end while
14:    end if
15:    $ES.Insert(\mu, h)$;
16: end while
17: return $\mathcal{N}$;
```

Fig. 5.2. The kNearestNeighbor procedure

The algorithm uses two auxiliary structures, namely the set S of features to be generated, and the set $\mathcal{N}$ of neighbor tuples to return as an answer. For convenience, tuples in $\mathcal{N}$ are sorted according to their distance from the query tuple μ. More in detail, lines 3-16 specify how the set $\mathcal{N}$ is filled. First, a feature x is extracted (line 4), and the *FI.Search* method is exploited to retrieve the logical address of the bucket associated with x. For each of these buckets, lines 9-13 iteratively extract the tuples it contain (using *ES.Read*) and try to insert them into $\mathcal{N}$. Specifically, a tuple ν can be inserted within $\mathcal{N}$ in two cases: *(i)* the size of $\mathcal{N}$ does not exceed its capacity, or *(ii)* $\mathcal{N}$ capacity is k, but it contains an element whose distance from μ is higher than the distance between ν and μ – actually, $\mathcal{N}.MaxDist()$ here denotes the maximum distance between μ and any tuple in $\mathcal{N}$. If needed, the element least similar to μ is removed from $\mathcal{N}$, in order to make room for ν. As a side effect, the algorithm updates FI and ES, in order to correctly refer to the novel tuple μ.

Hashing based on Exact Matching

A major point in kNearestNeighbor procedure is the choice of features for indexing tuples, which will strongly impact on the effectiveness and efficiency of the whole method, and should be carefully tailored to the criterion adopted for comparing tuples. To this purpose, we describe an indexing scheme which is meant to retrieve similar tuples, according to a set-based dissimilarity function, namely the *Jaccard distance*: for any two tuples $\mu, \nu \subseteq \mathcal{M}$, $dist_J(\mu, \nu) = 1 - |\mu \cap \nu| / |\mu \cup \nu|$. In practice, we assume that $dist_{\mathcal{M}}$ corresponds to the Dirichlet function, and that, consequently, the dissimilarity between two itemsets is measured by evaluating their degree of overlap.

In this case, a possible strategy for indexing a tuple μ simply consists in extracting a number of non-empty subsets of μ, named *subkeys* of μ, as indexing features. As the number of all subkeys for a given tuple is exponential in the cardinality of the tuple itself, the method was tuned to produce a minimal set of "significant" subkeys, which yet allow to retrieve all the tuples whose distance from μ is lower than a specified threshold δ.

In particular, a subkey s of μ is said *δ-significant* if $dist_J(\mu, s) \leq \delta$. Clearly enough, all the *δ-significant* subkeys of μ have size $|s|$ such that:

$$\lfloor |\mu| \times (1 - \delta) \rfloor \leq |s| \leq |\mu|$$

Notably, any tuple ν such that $dist_J(\mu, \nu) \leq \delta$ must contain at least one of the δ-significant subkeys of μ. Therefore, searching for tuples that exhibit at least one of the δ-significant subkeys derived from a tuple μ constitutes a strategy for retrieving all the neighbors of μ without scanning the whole database. Such a strategy also guarantees an adequate level of selectivity: indeed, if μ and ν contain a sensible number of different items, then their δ-significant subkeys do not overlap. As a consequence, the probability that μ is retrieved for comparison with ν is low.

In order to understand how the retrieving phase of our nearest neighbor approach works, we propose the following example.

A Simple Illustrative Example

Let consider the index structure exemplified in Figure 5.3. The index already contains the tuples $\mu = \{a, b, c, d\}$, $\nu = \{a, b, c, h\}$ and $\omega = \{a, c, f\}$, which are inserted by fixing $\delta = 0.1$ as distance threshold. It is easy to observe that each subkey is associated with a bucket containing the tuples sharing this subkey: for example, the bucket linked by the $\{a, b, c\}$ subkey contains both μ and ν, whereas the bucket linked by $\{a, c, f\}$ contains ω.

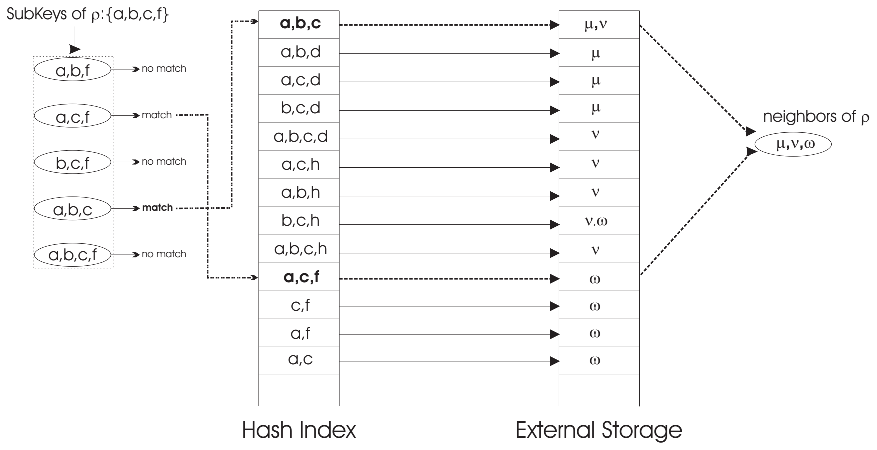


Fig. 5.3. Hash Index structure

When searching for the neighbors of a new tuple $\rho = \{a, b, c, f\}$, the algorithm generates the δ-significant subkeys of ρ, listed in the left of the figure. By querying the index structure with the subkeys, we obtain μ and ν (by means of $\{a, b, c\}$), and ω (by means of $\{a, c, f\}$).

5.5 Experimental Evaluation

In this section we study the behavior of the Generate-Clusters algorithm proposed in Figure 5.1. Experiments are aimed at evaluating whether the proposed indexing method allows substantial improvement in the clustering task, and whether an appropriate number of clusters is generated. In particular, we equip the proposed algorithm with the index structures studied in Section 5.4, and compare the results of the algorithm in the two cases. Since both indexes are stored on secondary memory, from a computational cost viewpoint it is important to evaluate the number of read and write operations as well. Thus, the efficiency of an index scheme is evaluated by three parameters:

- The number of distances computed during the selection of the neighbors. This is an effective evaluation parameter, which represents how many comparisons are performed during an insert/select operation and provides for an estimation of the CPU overhead.
- The number of disk pages read during the selection of objects. In principle, the hash-based approach could cause continuous leaps in the read operations, even if a small number of comparisons is needed.
- The number of disk pages written during the updating of the index. Since the index has to be incrementally maintained, it is important to evaluate the cost of such a maintenance.

We applied the algorithm to the task of de-duplicating tuples representing demographic information in a banking scenario. The instances of the algorithm were tested against a real-world data set consisting of 105,140 tuples, representing information about customers of an Italian bank. Data was firstly reconciled by exploiting the technique described in Chapter 3. Each tuple exhibited an average of 8 relevant neighbors, and in general distances exhibit high values.

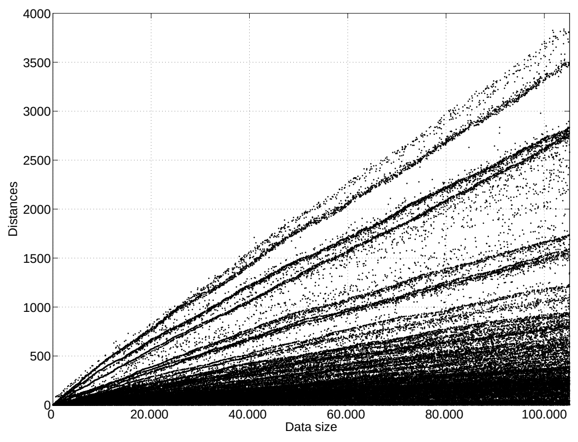
Figures 5.4-5.10 compare the performance of the clustering algorithm, equipped with both the M-Tree and the hash index structure described. We adopted the M-Tree implementation available on the Web.[1] The tree was tuned by setting a node size of 4K and a Random split policy. In both techniques we fixed $\delta = 0.2$ and $k = 10$.

The graphs represent the performances of the approaches w.r.t. the data size. In particular, the horizontal axis represents the portion of data examined thus far. The evaluation of the incremental behavior of the approach can be made by observing whether the increase of the measure under consideration is bounded. This clearly does not happen in Figures 5.4(a) and 5.5(a), representing the number of distances and I/O reads of the M-Tree approach respectively. In the average, the number of comparisons and I/O reads are low, as testified by the percentiles shown in Table 5.1. Nevertheless, a search in the M-Tree exhibits a substantially linear behavior in the number of objects stored in the tree. This is also testified by Figures 5.7 and 5.8. In these graphs, the performances of the approaches have been averaged on 5,000 tuples.

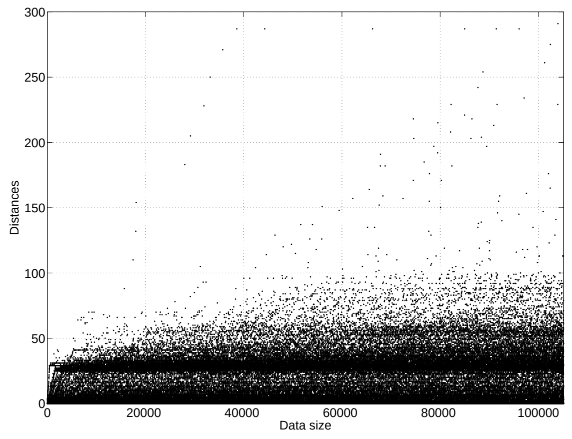
On the other side, the hash approach exhibits a performance which is bounded by a constant factor, as expected. In particular, 90% of the tuples retrieve their neighbors by exploiting less than 41 comparisons and 15 disk reads, as shown in Table 5.1. Also, graphs in Figures 5.7, 5.8, 5.9 and 5.10 show a substantial difference in the performance of the approach based on hashing w.r.t. the one based on the M-Tree.

An opposite trend can be seen in the number of disk writes (see Figure 5.6). This is mainly due to the different philosophy underlying the two structures. The number of disk writes, in the hash method, depends on the number of δ-relevant subkeys. The larger is the set of subkeys, the higher is the number

[1] Details can be found at `http://www-db.deis.unibo.it/Mtree/`.



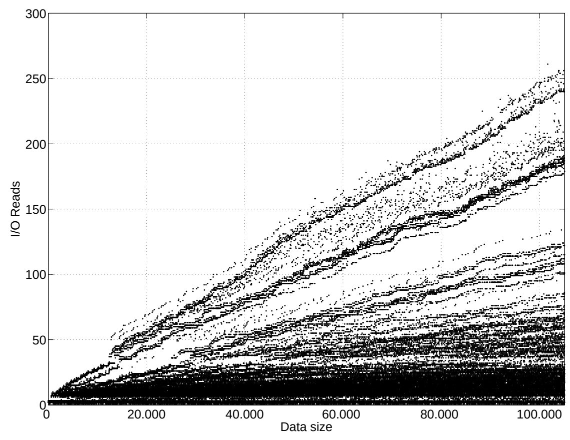
(a) M-Tree: No of distances computed per tuple



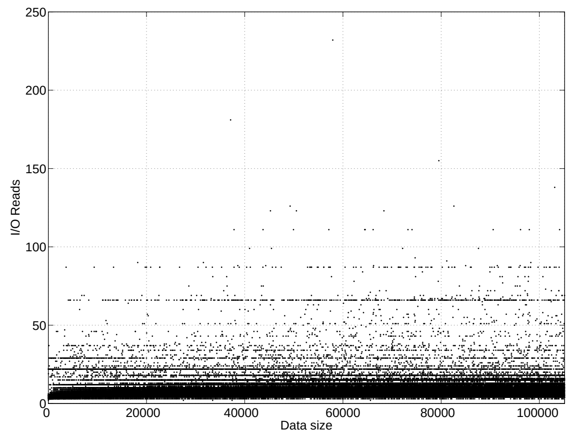
(b) Hash: No of distances computed per tuple

Fig. 5.4. Distances Performances

of writes needed in order to update the index. On the contrary, the M-Tree is a balanced structure whose update causes (at most) a number of writes which is proportional to the depth of the tree. Indeed, in order to update its structure, the M-Tree has to select the most appropriate position of the current tuple. After a suitable node has been selected (which does not necessarily correspond to the most suitable node), the tree inserts the tuple in the node and writes the node back to the disk. An overhead is possible only in case that the insertion



(a) M-Tree: No of IO Reads

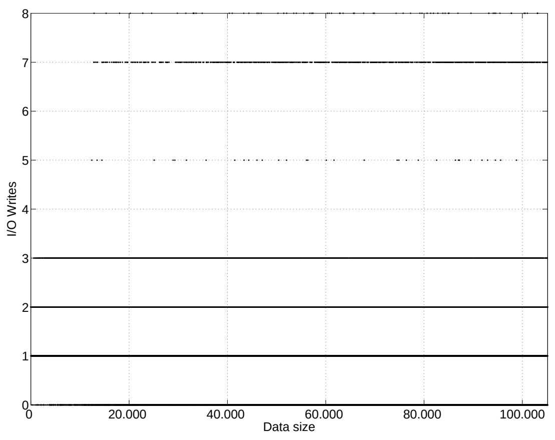


(b) Hash: No of IO Reads

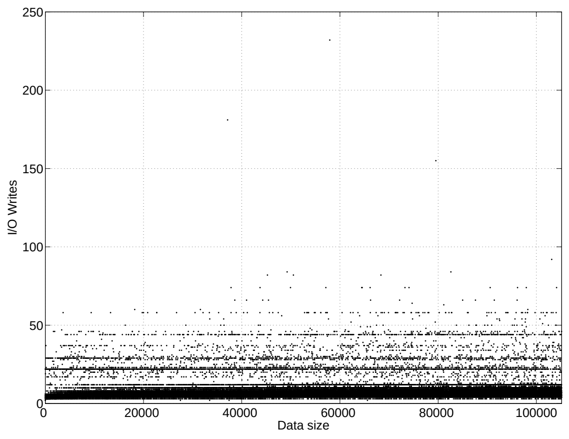
Fig. 5.5. IO Read Performances

causes a node overflow. In such a case, the node is split and the insertion is propagated upward in the tree.

Effectiveness can be evaluated by measuring the overlap between the expected number of clusters and the actual number of clusters computed. Clearly, the latter depends from the k and δ values. Since such values directly influence an indexing scheme in performing neighbor searches, the important issue is whether the neighbors retrieved from the index suffice to perform a correct classification. To this purpose, we introduce one measure,



(a) M-Tree: No of IO Writes



(b) Hash: No of IO Writes

Fig. 5.6. IO Write Performances

namely `FP-rate`. More precisely, given a tuple μ, and a database $\mathcal{DB}$, let *neighbors*$(\mu, \mathcal{DB})$ denote the set of relevant tuples for classifying μ, and let *retrieved*$(\mu, \mathcal{DB})$ denote the set of tuples actually retrieved when performing a search by using the index structure built for $\mathcal{DB}$. Then, `FP-rate`$(\mu, \mathcal{DB})$, represents the rate of (*False Positives*) tuples in $\mathcal{DB}$ which are retrieved but are not relevant for μ, i.e.:

$$\texttt{FP-rate}(\mu, \mathcal{DB}) = \mid retrieved(\mu, \mathcal{DB}) - neighbors(\mu, \mathcal{DB}) \mid$$

Distances	M-Tree	0	15	57	121	201	298	425	620	1013	1956	8719
	Hash	0	0	0	1	5	11	24	28	32	41	291
I/O Reads	M-Tree	1	2	8	12	16	22	30	47	74	139	633
	Hash	2	5	5	6	6	8	8	10	11	15	232
I/O Writes	M-Tree	0	0	0	1	1	1	1	1	2	2	12
	Hash	2	4	5	5	5	6	6	7	8	10	232

Table 5.1. Percentiles of the performances of the approaches

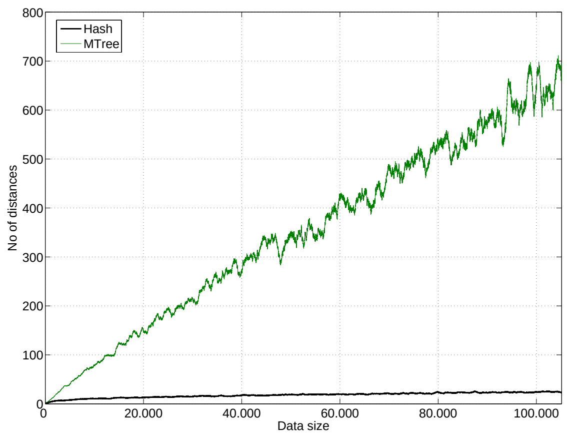


Fig. 5.7. No of distances on real data

The global `FP-rate` hence can be computed by averaging the `FP-rates` locally to each tuple μ_i and the pertaining database portion (i.e., the dataset $\{\mu_1, \ldots, \mu_{i-1}\}$).

Instead, we do not evaluate in this setting the `FN-rate` i.e., the rate of (*False Negatives*) tuples in $\mathcal{DB}$, which are relevant for a tuple but are not retrieved. Indeed, the proposed indexing method has been studied to work in combination with Jaccard similarities, and hence does not provide false negatives. In Chapter 6 we will treat how the method can be modified in order to deal with different dissimilarity measures which in principle could report false negatives.

Effectiveness was measured over synthetic data. Here, tuples were generated according to the following parameters:

- the average size T of itemsets constituting the tuples.
- the size of $\mathcal{M}$.

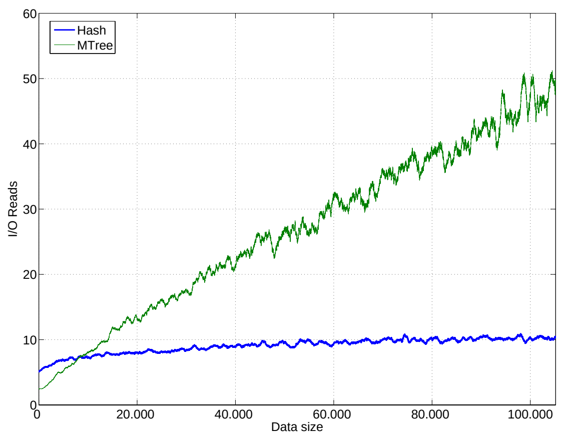


Fig. 5.8. No of I/O Reads on real data

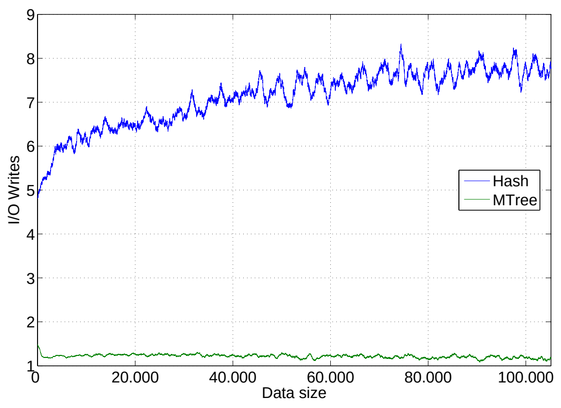


Fig. 5.9. No of I/O Writes on real data

- the number of clusters C.
- the number of tuples N.

More in detail, each cluster was generated by randomly choosing a subset of the items in $\mathcal{M}$. Then, each tuple in the cluster was generated by choosing items from the subset associated with the cluster. In order to guarantee the right degree of overlap, each new tuple was generated as a variation of a previously generated one.

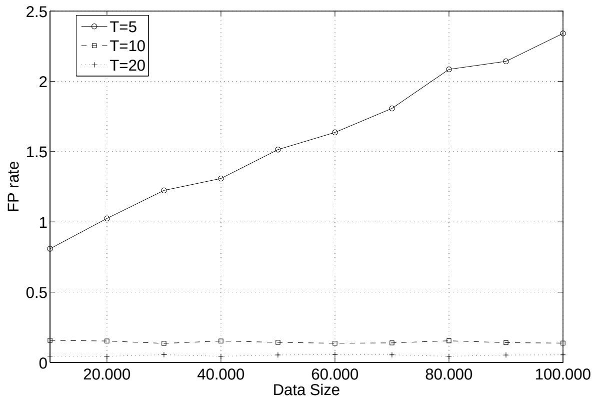


Fig. 5.10. `FP-rate` vs. data size on synthetic data

The synthetic data sets, used to test the effectiveness of the hashing approach, was generated according to the following parameters: $T = [5, 10, 20]$, $N = 100,000$, $C = 20,000$, and $\mathcal{M} = 400,000$.

We tested the hash-based approach for increasing values of T. Figure 5.10 summarizes the values of `FP-rate`. Notice that, the rate is constant (fairly low) except in the case $T = 5$. The latter exhibits higher values mainly because the size of the itemsets contained in the tuples causes the generation of 1-subkeys, which cause a large number of false positives.

5.6 Qualitative Comparison

In the following, we shortly perform a qualitative comparison of our approach with some relevant proposals for the detection and management of duplicated data already reviewed in Chapter 4. As there discussed, this problem has given rise to a large body of work in several research communities, and under a variety of names (such as, e.g., Merge/Purge, Record Linkage, De-duplication, Entity-Name Matching, Object Identification).

In most of these approaches, a central issue is the definition of a method for comparing objects, especially when information on object identity is carried by textual fields (indeed, the latter are subject to various kinds of heterogeneity and mismatches across different information sources). To this purpose, in addition to classical string (dis)similarity functions [71], several methods [103, 104, 36, 12, 125] were defined, which allow to effectively compare textual information in the context of duplicated data.

Many approaches to the de-duplication problem essentially attempt to match or cluster duplicated records [37, 103], based on suitable similarity functions. Unfortunately, most of these approaches mainly focus on effectiveness issues, while paying minor attention to scalability, and end up being inadequate under stronger efficiency requirements. It is worth noticing that resorting to consolidated clustering algorithms [73, 48, 69, 68, 58], could not guarantee an adequate level of scalability either. Indeed, even these algorithms would not work adequately in a situation where far too many clusters are expected to be found, as it does happen in a typical de-duplication scenario, where the number of clusters can be of the same order as the size of the database. To the best of our knowledge, the only suitable approach appear to be the one proposed in [98]. Here, the authors avoid costly pairwise comparisons by grouping objects in "canopies", i.e., subsets containing objects suspected to be similar according to a given similarity function, and then computing pairwise similarities only within 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 in the approach is the creation of canopies. The authors proposed an effective solution to this issue: nevertheless the approach they propose does not cope with incrementality issues. In a sense, our approach also builds canopies (which are collected within the same buckets in the index), the main difference being that our approach allows to approximately detect such canopies incrementally.

There is a plenty of approaches for distance-based search in metric spaces (see, e.g., [26, 75] for a survey). Again, these approaches suffer from the high dimensionality of the space where search is performed, as described in [150]: indeed, high dimensionality causes too sparse regions to analyze, and thus invalidate the proposed index methods.

In Section 5.4.1 we faced the above phenomenon by introducing the *M-tree* index structure [30]. The M-tree is a paradigmatic example, being particularly suitable to perform searches in general metric spaces (and even in spaces equipped with edit-distance). However, while in "dense" domains it allows to simultaneously reduce the number of distance evaluations and the amount of I/O performed, its performances are expected to degrade when applied to a typical de-duplication application.

Recently, some approaches have been proposed [65, 5, 24] which exploit efficient indexing schemes based on the extraction of relevant features from the tuples under consideration. Such approaches could be adapted to deal with the problem of de-duplication, even though they are not specifically designed to approach the problem from an incremental clustering perspective, as we instead did here.

5.7 Conclusions

In this chapter, we addressed the problem of recognizing duplicate information, specifically focusing on scalability and incrementality issues. The core of the proposed approach is an incremental clustering algorithm, which aims at discovering clusters of duplicate tuples, based on a novel hashing-based indexing technique. An empirical analysis, both on synthesized and real data, showed the validity of the approach, which does exhibit a considerable improvement in performance with respect to a traditional, state-of-the- art, index structure (M-tree).

The proposed approach is expected to efficiently recognizing duplicate tuples which are similar according to set-based similarity functions. In particular, the effectiveness of the approach is based on the adoption of the Jaccard distance, which does not exploit the dissimilarity $dist_{\mathcal{M}}$ between tokens. Experiments on real data, show that the above dissimilarity works well in practical cases. Nevertheless, the approach could in principle fail in cases where a more refined dissimilarity functions is needed [12, 36].

To this purpose, in Chapter 6 we introduce a more refined key-generation technique for the algorithm in Figure 5.3, which allows a controlled level of approximation in the search for the nearest neighbors of a tuple. More in detail, we will exploit the family of *Locality-Sensitive hashing functions* (see Section 4.5.3), which are guaranteed to assign two any objects to the same buckets with a probability which is directly related to their degree of mutual similarity.

6

Incremental Clustering with Improved Hashing

6.1 Introduction

In Chapter 5, we proposed some basic ideas to cope with the duplicate detection problem from an incremental perspective. Specifically, we proposed an indexing scheme tailored to a set-based dissimilarity function, where each tuple was regarded as sets of tokens, and a number of relevant subsets were exploited for indexing it. For the sake of simplicity, in this chapter we also refer the previous hash-based method as *naïve hashing* approach.

Despite its simplicity, the naïve indexing scheme was proven to work quite well in practical cases (see Section 5.6). Notwithstanding, two main drawbacks can be observed:

1. The cost of the approach critically depends on the number of δ-*significant* subkeys: the larger is the set of subkeys, the higher is the number of writes needed to update the index.
2. More importantly, the proposed key-generation technique suffers from a coarse grain dissimilarity between itemsets, which does not take into account a more refined definition of $dist_{\mathcal{M}}$. Indeed, the proposed approach is subject to fail, in principle, in cases where likeliness among single tokens are to be recognized as well. As an example, the tuples:

μ_1	`Jeff, Lynch, Maverick, Road, 181, Woodstock`
μ_2	`Jef, Lync, Maverik, Rd, 181, Woodstock`

   are not recognized as similar in the naïve approach (even though they clearly refer to the same entity), due mainly to a dissimilarity between single tokens which is not kept by a simple matching between tokens. Notice that, lowering the degree δ of dissimilarity, partially alleviates such an effect, but at the cost of worsening the performances of the index considerably.

In this chapter, we extend and improve the proposal in Chapter 5 both in effectiveness and efficiency. First, we gain a direct control over the number of features used for indexing any tuple, which is a major parameter that critically impacts on the overall cost of the approach. Moreover, we tune the approach to be less sensitive to little differences between matching tokens.

In practice, we exploit a refined key-generation technique which, for each tuple under consideration, guarantees a controlled level of approximation in the search for the nearest neighbors of such a tuple. To this purpose, we resort to a family of *Locality-Sensitive hashing* functions [77, 20, 19, 62], which are guaranteed to assign any two objects to the same buckets with a probability which is directly related to their degree of mutual similarity.

The rest of this chapter is organized as follows. In Section 6.2, the new key-generation technique is proposed. In Section 6.3 an extensive evaluation about the effectiveness of the proposed approach is performed. Finally, in Section 6.4 some considerations are drawn.

6.2 Hierarchical Approximate Hashing based on *q*-grams

Our objective in this section is to define a hash-based index which is capable of overcoming the above described drawbacks. In particular, we aim at defining a key-generation scheme which allows a constant number of disk writes and reads, being simultaneously capable of keeping a fixed (low) rate of false negatives.

To overcome these limitations, we have to generate a fixed number of sub-keys, which, however, are capable of reflecting both the differences among itemsets, and those among tokens. To this purpose, we define a key-generation scheme by combining two different techniques:

- the adoption of hash functions based on the notion of *minwise independent permutation* [62, 19], for bounding the probability of collisions.

- the use of q-grams (i.e., contiguous substrings of size q) for a proper approximation of the similarity among string tokens [65].

As introduced in Section 4.5.3, a *locally sensitive* hash function H for a set S equipped with a distance function D, is a function which bounds the probability of collisions to the distance between elements. Formally, given H, for each pair $p, q \in S$ and value ϵ, there exists values P_1^ϵ and P_2^ϵ such that:

- if $D(p,q) \leq \epsilon$ then $\Pr[H(p) = H(q)] \geq P_1^\epsilon$, and
- if $D(p,q) > \epsilon$ then $\Pr(H(p) = H(q)] > P_2^\epsilon$.

Clearly, such a function H provides a simple solution to the problem of false negatives described in the previous chapter. Indeed, for each μ, we can define a representation $rep(\mu) = \{H(a) | a \in \mu\}$, and fill the hash-based index by exploiting δ-significant subkeys from such a representation.

To this purpose, we can exploit the theory of *minwise independent permutations* [19]. Let recall that, a minwise independent permutation is a coding function π of a set X of generic items such that, for each $x \in X$, the probability of the code associated with x being the minimum is uniformly distributed, i.e.:

$$\Pr[\min(\pi(X)) = \pi(x)] = \frac{1}{|X|}$$

A minwise independent permutation π naturally defines a locally sensitive hash function H over an itemset X, defined as $H(X) = \min(\pi(x))$. Indeed, for each two itemsets X and Y, it can be easily verified that

$$\Pr[\min(\pi(X)) = \min(\pi(Y))] = \frac{|X \cap Y|}{|X \cup Y|}$$

This suggests that, by approximating $dist_{\mathcal{M}}(a_i, a_j)$ with the Jaccard similarity among some given features of a_i and a_j, we can adopt the above envisaged solution to the problem of false negatives. When $\mathcal{M}$ contains string tokens (as it usually happens in a typical entity resolution setting), the features of interest of a given token a can be represented by the q-grams of a. It has been shown in [65, 145] that the comparison of the q-grams provides a suitable approximation of the Edit distance, which is typically adopted as a classical tool for comparing strings.

In the following, we show how minwise functions can be effectively exploited to generate suitable keys. Let consider the example tuples:

μ_1	`Jeff, Lynch, Maverick, Road, 181, Woodstock`
μ_2	`Jef, Lync, Maverik, Rd, Woodstock`

Clearly, the tuples μ_1 and μ_2 refer to the same entity (and hence should be associated with the same key). The detection of such a similarity can be accomplished by resorting to the following observations:

1. some tokens in μ_1 are strongly similar to tokens in μ_2. In particular, `Jeff` and `Jef`, `Lynch` and `Lync`, `Road` and `Rd`, and `Maverick` and `Maverik`. Thus, the tuples can be represented as:

μ_1	$a_1, a_2, a_3, a_4,$ `181,Woodstock`
μ_2	$a_1, a_2, a_3, a_4,$ `Woodstock`

   where a_1, a_2, a_3, and a_4 represent the four "approximately common" terms.
2. the postprocessed tuples exhibit only a single mismatch. If a minwise permutation is applied to both, with high probability the resulting key shall be the same.

Thus, a minwise function can be applied over the "purged" representation of a tuple μ, in order to obtain an effective key. The purged version of μ should guarantee that tokens exhibiting high similarity with tokens in other tuples, change their representation towards a "common approximate" token.

Again, the approximate representation of a token, described in point 1 of the example above, can be obtained by resorting to minwise functions. Given two tokens a_i and a_j, recall that their dissimilarity $d_{\mathcal{M}}(a_i, a_j)$ is defined in terms of the Jaccard coefficient. In practice, if $\mathit{feat}(a)$ represents a set of features of token a, then:

$$d_{\mathcal{M}}(a_i, a_j) = 1 - \frac{|\mathit{feat}(a_i) \cap \mathit{feat}(a_j)|}{|\mathit{feat}(a_i) \cup \mathit{feat}(a_j)|}$$

The set $\mathit{feat}(a)$ can be defined in terms of q-grams. The latter represent the simplest yet effective information contents of a token and indeed, they have been widely used and demonstrated fruitful in estimating the similarity of strings [65, 145]. Hence, by applying a minwise function to the set of q-grams of a, we again have the guarantee that similar tokens collapse to a unique representation.

Thus, given a tuple μ to be encoded, the key-generation scheme we propose works in two different hierarchical levels:

- In the first level, each element $a \in \mu$ is encoded by exploiting a minwise hash function H^l. This guarantees that two similar but different tokens a and b are with high probability associated with a same code. As a side effect, tuples μ and ν sharing "almost similar" tokens are purged into two representations where such tokens converge towards unique representations.
- In the second level, the set $\mathit{rep}(\mu)$ obtained from the first level is encoded by exploiting a further minwise hash function H^u. Again, this guarantees that purged tuples sharing several codes are associated with a same key.

The key resulting from the final, second-level coding can be effectively adopted in the indexing structure described in Section 5.4.2.

A key point is the definition of a proper family of minwise independent permutations upon which to define the hash functions. A very simple idea is to randomly map a feature x of a generic set X to a natural number. Then, provided that the mapping is truly random, the resulting probability of mapping a generic $x \in X$ to a minimum number is uniformly distributed, as required. However, in practice, it is hard to obtain a truly random mapping. Hence, we exploit a family of "practically" minwise independent permutations [19], i.e., the functions:

$$\pi(x) = (a \cdot c(x) + b) \bmod p$$

where $a \neq 0$ and $c(x)$ is a unique numeric code associated with x (such as, e.g. the code obtained by the concatenation of the ASCII characters it includes).

Provided that a, b, $c(x)$ and p are sufficiently large, the behavior of π is practically random, as we expect.

We further act on the randomness of the encoding, by combining several alternative functions (obtained choosing different values of a, b and p) as shown in Figure 6.1.

Recall that a hash function on π is defined as $H_\pi(X) = \min(\pi(X))$, and that $\Pr[H_\pi(X) = H_\pi(Y)] = |X \cap Y|/|X \cup Y| = \epsilon$. Notice that the choice of a, b and p in π introduces a probabilistic bias in H_π, which can in principle leverage false negatives.

Let us consider the events $A \equiv$"sets X and Y are associated with the same code", and $B = \neg A$. Then, $p_A = \epsilon$ and $p_B = 1 - \epsilon$. By exploiting h different encodings $H_1^l, \ldots, H_h^l$ (which differ in the underlying π permutations), the probability that all the encodings exhibit a different code for X and Y is $(1-\epsilon)^h$. If $\epsilon > 1/2$ represents the average similarity of items, we can exploit the h different encodings for computing h alternative representations $rep_1(\mu), \ldots, rep_h(\mu)$ of a tuple μ. Then, by exploiting all these representations in a disjunctive manner, we lower the probability of false negatives to $(1-\epsilon)^h$.

```
HASH($\mu = \{a_1, \ldots, a_n\}$, $k$, $h$, $q$)
1:  for each $a_i \in \mu$ do
2:      compute the $q$-gram representation $q_i$ of $a_i$;
3:      compute $h$ encodings $H_1^l(q_i), \ldots H_h^l(q_i)$;
4:  end for
5:  for $i = 1$ to $h$ do
6:      $rep_i(\mu) \leftarrow \{H_i^l(q_j) | a_j \in \mu\}$;
7:      for $j = 1$ to $k$ do
8:          $e_i^j \leftarrow H_j^u(rep_i(\mu))$;
9:      end for
10:     $key_i \leftarrow e_i^1 \wedge e_i^2 \wedge \ldots \wedge e_i^k$;
11: end for
12: return $\{key_1, \ldots, key_h\}$;
```

Fig. 6.1. The key-generation procedure

In general, allowing several trials generally flavors high probabilities. Consider the case where $\epsilon < 1/2$. Then, the probability that, in k trials (corresponding to k different choices of a, b and p) at least one trial is B, is $1 - \epsilon^k$. We can apply this to the second-level encoding, where, conversely from the previous case, the probabilistic bias can influence false positives. Indeed, two dissimilar tuples μ and ν could in principle be associated with the same token, due to a specific bias in π which affects the computation of minimum random code both in $rep_i(\mu)$ and in $rep_i(\nu)$. If, by the converse a key is computed as a concatenation of k different encodings $H_1^u, \ldots, H_k^u$, the probability of having a different key for μ and ν is $1 - \epsilon^k$, where ϵ is the Jaccard similarity between $rep_i(\mu)$ and $rep_i(\nu)$.

6.3 Experimental Evaluation

The discussion in the previous section shows that the effectiveness of the approach relies on proper values of h and k. Essentially, low values of h leverage false negatives, whereas high values leverage false positives. Analogously, low values of k leverage false positives, whereas high values should, in principle, leverage false negatives.

Thus, this section is devoted to study suitable values of these parameters that fix a high correspondence between the retrieved and the expected neighbors of a tuple. To this purpose, for a generic tuple μ we are interested in evaluating some well-known measures as:

- the number TP_μ of *true positives* (i.e., the tuples which are retrieved and that belong to the same cluster of μ)

,and compare it to:

- the number of *false positives* FP_μ (i.e., tuples retrieved without being neighbors of μ).
- the number of *false negatives* FN_μ (i.e., neighbors of μ which are not retrieved).

Moreover, as global indicators we exploit the average precision and recall per tuple:

$$precision = \frac{1}{N} \sum_{\mu \in \mathcal{DB}} \frac{TP_\mu}{TP_\mu + FP_\mu}$$
$$recall = \frac{1}{N} \sum_{\mu \in \mathcal{DB}} \frac{TP_\mu}{TP_\mu + FN_\mu}$$

where N denotes the number of tuples in $\mathcal{DB}$.

The values of such quality indicators influence the effectiveness of the clustering scheme in Figure 5.1. In general, high values of precision allows for correct de-duplication: indeed, the retrieval of true positives directly influences the MostLikelyClass procedure which assigns each tuple to a cluster.

When precision is low, the clustering method can be effective only if recall is high.

Notice that low precision may cause a degradation of performances, if the number of false positives is not bounded. Thus, we also evaluate the efficiency of the indexing scheme, in terms of the number of tuples retrieved by each search. This value depends on h and k, and is clearly related to the rate of false positives.

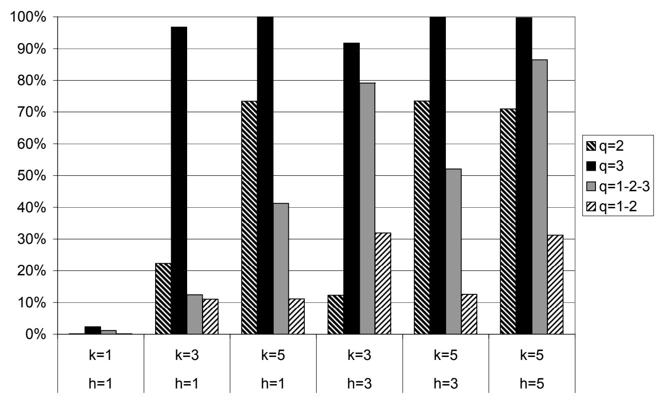
Experiments were conducted on both real and synthesized data. For the real data, we newly exploited the collection of about 105,140 tuples, representing information about customers of an Italian bank. Synthetic data was produced, according to the parameters described in Section 5.5, by generating $C = 50,000$ clusters, $N = 1,000,000$ tuples (i.e., an average of 20 tuples per cluster), and each tuple containing $T = 20$ tokens in the average.

Each cluster was obtained by first generating a representative of the cluster, and then producing the desired duplicates as perturbations of the representative. The perturbation was accomplished either by deleting, adding or modifying a token from the cluster representative. The number of perturbations was governed by a *gaussian distribution* having p as mean value. This further parameter p was exploited to study the sensitivity of the proposed approach to the level of noise affecting the data, and due, for example to misspelling errors. It is worth noticing that, since the naïve hashing approach is insensitive to the dissimilarity between single tokens, then in Section 5.5 we do not consider the perturbation percentage p (or, analogously we set $p = 0$).

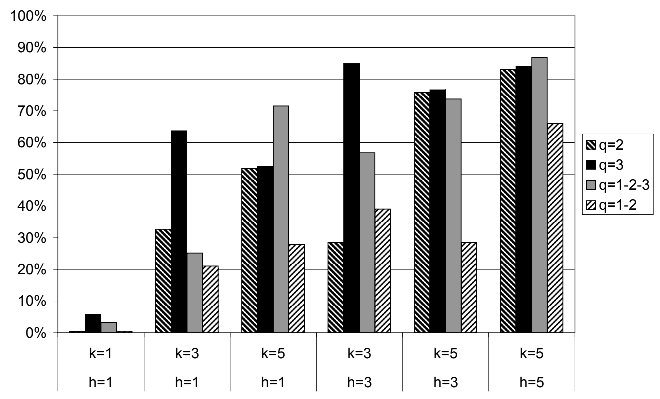
Figures 6.2 and 6.3 illustrate results of some tests we conducted on this synthesized data, in order to analyze the sensitivity of the retrieval to the parameters q, h and k (relative to the indexing scheme), and p (relative to the noise in the data). In particular, the values of q ranged over 2, 3, 1-2 (both 1-grams and 2-grams) and 1-2-3 (q-grams with sizes 1, 2 and 3).

Figures 6.2(a) and (b) show the results of precision and recall for different values of h and k, and $p = 2$. We can notice that precision raises on increasing values of k, and decreases on increasing values of h. The latter statement does not hold when q-grams of size 1 are considered. In general, stabler results are guaranteed by using q-grams of size 3. As to the recall, we can notice that, when k is fixed, increasing values of h correspond to improvements as well. If h is fixed and k is increased, the recall decreases only when $q = 3$. Here, the best results are guaranteed by fixing q=1-2-3. In general, when $h \geq 3$ and $k \geq 3$, the indexing scheme exhibits good performances.

Figures 6.3(a) and (b) are useful to check the robustness of the index. As expected, the effectiveness of the approach tends to degrade when higher values of the perturbation factor p are used to increase intra-cluster dissimilarity. However, the proposed retrieval strategy keeps on exhibiting values of precision and recall that can still enable an effective clustering. In more detail, the impact of perturbation on precision is clearly emphasized when tuples are encoded by using also 1-grams, whereas using only either 2-grams or 3-grams allows for making precision results stabler. Notice that for $q = 3$ a nearly



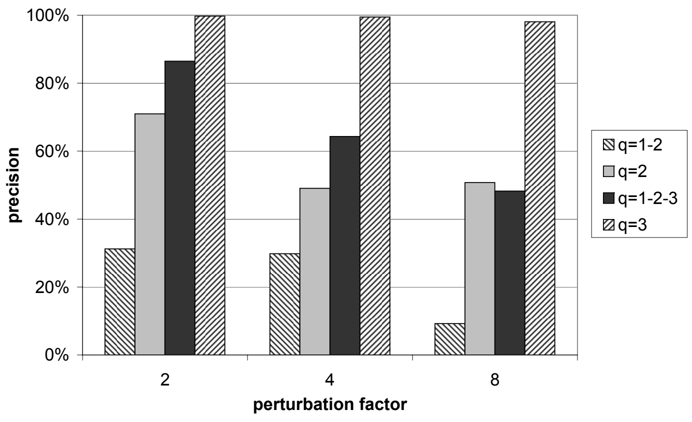
(a) precision vs. h, k and q



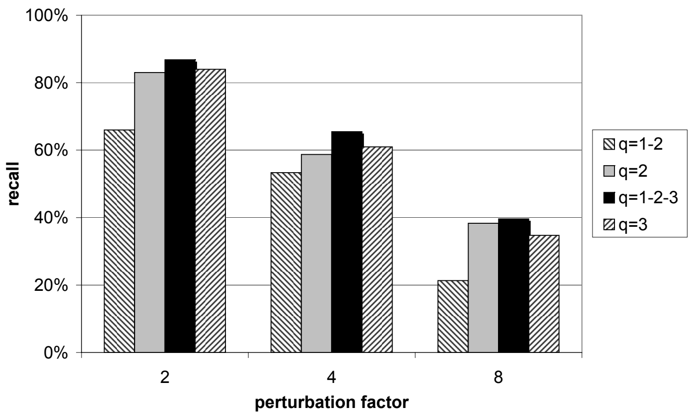
(b) recall vs. h, k and q

Fig. 6.2. Results on synthetic data w.r.t. q-gram size (q) and nr. of hash functions (h,k)

maximum value of precision is achieved, even when a quite perturbed data set is used.



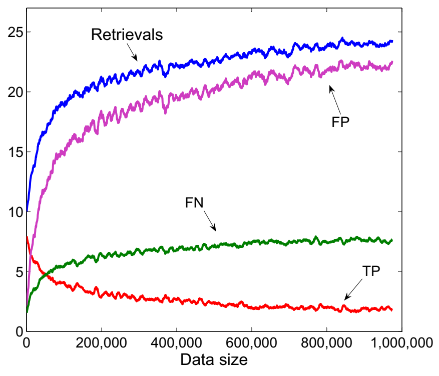
(a) precision vs. perturbation and q



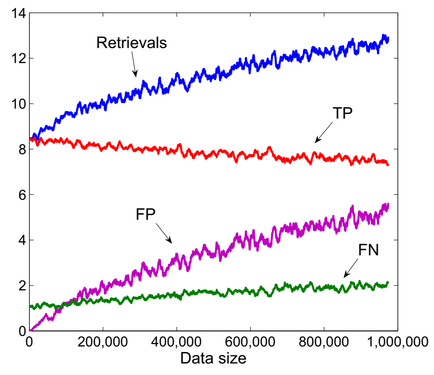
(b) recall vs. perturbation and q

Fig. 6.3. Results on synthetic data w.r.t. q-gram size (q) and perturbation

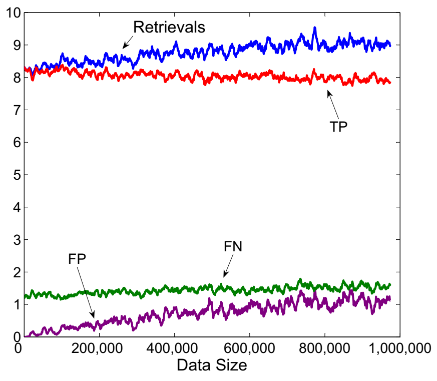
Figure 6.4 provides some details on the progress of the number of retrieved neighbors (indicated as *Retrievals*), TP, FP, and FN when an increasing number of tuples, up to 1,000,000, is inserted in the index. For space reasons, only some selected combinations of h and k, and q are considered, which were deemed as quite effective in previous analysis. Anyway, we pinpoint that some general results of the analysis illustrated here also apply to other cases. Let observe that the values plotted in Figure 6.4 are averaged on a window of 5,000 tuples.



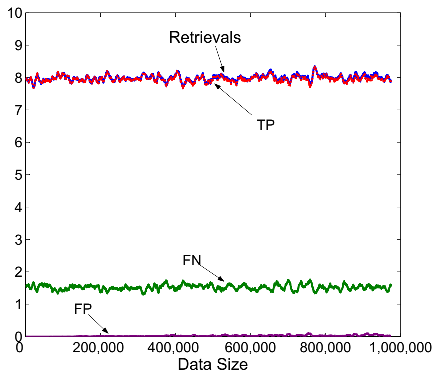
(a) $q = 2, h = 3, k = 3$



(b) $q = 2, h = 5, k = 5$



(c) $q = 3, h = 3, k = 3$



(d) $q = 3, h = 5, k = 5$

Fig. 6.4. Scalability w.r.t. the data size

In general, it is interesting to observe that the number of retrievals for each tuple is always bounded, although for increasing values of the data size the index grows. This general behavior, which we verified for all configurations of h, k and q, clearly demonstrates the scalability of the approach. In particular, we observe that for $q = 3$ the number of retrievals is always very low and nearly independent of the number of tuples inserted (see Figures 6.4(c) and (d)). More in general, the figures confirm the conceptual analysis that the number of I/O operations directly depends on the parameter h, the latter determining the number of searches and updates against the index.

All these figures also agree with the main outcomes of the analysis on effectiveness we previously conducted with the help of Figure 6.2. In particular, notice the quick decrease of FP and FN when both k and h turn from 3 to 5, in the case of $q = 2$ (Figure 6.4(a) and (b)), that motivates the improvement in both precision and recall observed in these cases. Moreover, the high precision guaranteed by using our approach with q-grams of size 3, is substantiated by Figures 6.4(c) and (d), where the number of retrieved tuples is very close to TP; in particular, for $k = 5$ (Figure 6.4(d)) the FP curve definitely flatten on the horizontal axis.

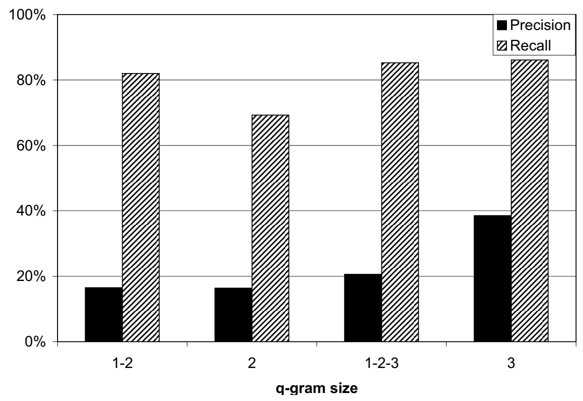
The above considerations are confirmed by experiments on real data. Figure 6.5(a) shows the results obtained for precision and recall by using different values of q, whereas figure 6.5(b) summarizes the average number of retrievals and quality indices. As we can see, recall is quite high even if precision is low (thus allowing for a still effective clustering). Notice that the average number of retrievals is low, thus guaranteeing a good scalability of the approach.

6.4 Conclusions

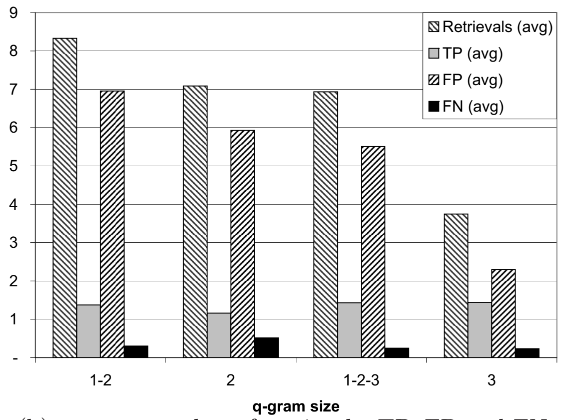
In this chapter, we tackle again the problem of recognizing duplicate information in a scenario where scalability and incrementality issues are not negligible. The approach is still based on the incremental clustering algorithm analyzed in Section 5.3, which aims at discovering clusters of duplicate tuples. However, here we propose a more refined hashing technique, which, by overcoming the drawbacks showed by our previous approach (see Section 6.1), it allows a controlled level of approximation in the search for the nearest neighbors of a tuple. In this novel hashing approach, the key-generation technique exploits the theory of *locality sensitive hashing* functions (LSH).

It is worth noticing that, the theory of LSH in not new in literature [62, 19, 20], and its application to the retrieval of nearest neighbors in high-dimensional spaces has been thoroughly studied: [62, 61], in particular, adopt a hierarchical combination of LSH functions similar to ours. However, their objective is rather different: given a distance d, an object q and a threshold δ, retrieve the set $S = \{p \mid d(q, p) < \delta\}$ in $O(|S|)$. By contrast, in a de-duplication scenario the value δ is unknown, and is in general parametric to the similarity function to be adopted. This also explains why the efforts, in the current literature, mainly concentrate on the definition of a proper distance function.

Thus, applying LSH in this context raises some novel, specific issues, which have not yet been studied. In particular, since the level of mismatch (e.g., mispelling errors) actually affecting duplicate tuples is an application-dependent



(a) precision and recall



(b) average number of retrievals, TP, FP and FN

Fig. 6.5. Results on real data using different q-gram sizes

parameter, it is not clear how to tune the h and k parameters governing our hierarchical approach. On the other hand, the extraction of q-grams from input strings introduces a further degree of freedom in the method, since determining a proper value for q-gram sizes is not a trivial task, in general.

Notice that our experimental evaluation in this respect is crucial, as it allows to get insight on the issue of optimally setting the above parameters. Thus, the experimental work showed in Section 6.3 constitutes a relevant, yet novel, contribution in itself. However, the way to optimally tune these parameters is still an open and interesting problem to further investigate.

The proposed approach has showed a quite effective behavior when the available information is based solely on strings, and Jaccard similarity is adopted to compare the features two tuples exhibit. We remark here that the described technique does not consider the database schema, whose adoption (and the consequent separation of the available string into fields) would likely allow to obtain a more refined de-duplication strategy: as an example, two tuples with the same “name” are more likely to be duplicates than two tuples with the same “city”. Nevertheless, the approach we presented here can be easily and effectively adapted to such a situation as well: the overall dissimilarity among tuples can be expressed as a combination of the dissimilarities relative to single fields, and consequently the retrieval of similar tuples can be accomplished by combining the keys relative to different fields and exploiting them within the index.

Clearly, when strings are too small or too different to contain enough informative content, the de-duplication task cannot be properly accomplished by the proposed clustering algorithm. To this purpose, as further direction, can be thought to study the extension of the proposed approach to different scenarios, where more informative similarity functions can be exploited. An example is the adoption of link-based similarity: recently, some techniques were proposed [82] which have been proved effective but still suffer from the incrementality issues which are the focus of our approach.

7

Conclusions

7.1 Summary of the Results

Identifying distinct records that refer to the same entity is an important information integration problem that has been studied for many years across several research communities, and known with the name (amongst others) of *Entity Resolution*. More in detail, Entity Resolution indicates a complex, multi-steps process embracing two main tasks: *Schema Reconciliation* and *Data Reconciliation*, aiming at addressing the structural and lexical heterogeneities respectively.

This thesis focused on Entity Resolution and its main contribution consisted in a systematic study of this overall process and the issues related with the severe constraints occurring when large volumes of data are involved. In the development of this thesis we followed three main directions:

1. We started by facing the preparatory Schema Reconciliation step as a typical Information Extraction problem of segmenting information in free text into a fixed attribute schema. Hence, we analyzed some state-of-the-art proposals in literature. In particular, we treated three stochastic classification approaches representing "global classification models" in the sense they ever tended to assign a class label to each term of text sequence under consideration, and hence they were unable to prevent the problem of misclassifying unknown tokens characterizing typical Entity Resolution settings. We overcame such a drawback by introducing *RecBoost*, a supervised "local" rule-based classifier that, in conjunction with a multi-stage classification scheme, it only handled the local specificities that was able to cope with, by postponing the unknown cases to subsequent classification stages. The effectiveness of our approach was demonstrated in an intensive experimental analysis where *RecBoost* was compared against some well-known text segmentation systems.
2. We approached the Data Reconciliation task as the problem of discovering duplicate tuples in a very large database from a clustering perspective:

given a set of tuples, our objective was to recognize subsets (clusters) of tuples such that intra-cluster similarity was high, and inter-cluster similarity was low. To this purpose, we presented an incremental algorithm based on a *k–Nearest Neighbor* schema. The core of this approach was the usage of a suitable indexing technique which, for any newly arrived tuple, allowed to efficiently retrieve a set of tuples in the database which were likely mostly similar, and hence were expected to refer to the same real-world entity. The proposed indexing technique was based on a hashing scheme, which tended to efficiently recognizing duplicate tuples according to the set-based *Jaccard* similarity function. Experiments, both on synthesized and real data, showed the validity of the approach which exhibited a considerable improvement in performance with respect to the *M-tree*, a traditional index structure able to perform searches in metric spaces.

3. We presented an extension of the previous proposal by allowing for a direct control on the degree of granularity needed to properly discover the actual neighbors (duplicates) of a tuple. To this purpose, we exploited a new key-generation approach, hierarchically obtained by combining two different techniques: *(i)* the adoption of hash functions based on the notion of *min-wise independent permutation* for bounding the probability of collisions, and *(ii)* the use of *q-grams* for a proper approximation of the similarity among string tokens in the tuples. By resorting to the min-wise independent permutations, we were guaranteed to assign any two tuples to a same bucket with a probability which was directly related to their degree of mutual similarity. Finally, a detailed evaluation analysis demonstrated, in both effectiveness and efficiency, the improvement respect to our first approach.

Several are the novelties introduced by our proposals respect to the classical approaches in literature, both for schema and data reconciliation. Some (main) differences can be summarized as follows:

- *RecBoost* pursued a new methodological approach in which a strict cooperation between ontology-based generalization and rule-based classification was envisaged, which allowed to reliably associate terms in a free text with a corresponding attribute in a fixed descriptor. The key feature of our approach was the introduction of *progressive classification*, which iteratively enriched the available ontology, thus allowing to incrementally achieve accurate schema reconciliation. This ultimately differentiated our approach from previous works in the current literature (e.g., [15, 1]), which adopted schemes with fixed background knowledge, and hence unlikely able to capture the multi-faceted peculiarities of the data under investigation.
- As our proposal for de-duplicating a dataset, other approaches in literature attempted to match or cluster duplicated records [37, 103] based on suitable similarity functions. However, most of these approaches mainly focused on effectiveness issues by paying minor attention to scalability, thus

ended up being inadequate under stronger efficiency requirements. Actually, we founded an adequate approach in [98]. Here, the authors avoided costly pairwise comparisons by grouping objects in "canopies" (i.e., subsets containing objects suspected to be similar according to a given similarity function), and then computing pairwise similarities only within canopies. However, the approach they proposed was unable to cope with incrementality issues as our approach, instead, has done.

- As stated in the Chapter 6, also the theory of LSH was not new in literature [62, 19, 20], and its application to the retrieval of nearest neighbors in high-dimensional spaces has been thoroughly studied. In particular, in [62, 61], the authors adopted a hierarchical combination of LSH functions similar to ours. However, their objective was rather different: given a distance d, an object q and a threshold δ, retrieve the set $S = \{p \mid d(q,p) < \delta\}$ in $O(|S|)$. By contrast, in a de-duplication scenario the value δ was unknown, and was in general parametric to the similarity function to be adopted. This also explained why the efforts in the current literature mainly were concentrated on the definition of a proper distance function. Thus, by applying LSH in this context raised some novel, specific issues, which have not yet been studied. In particular, since the level of mismatch (e.g., mispelling errors) actually affecting duplicate tuples is an application-dependent parameter, was unclear how to tune the parameters governing our hierarchical approach. The related experimental evaluation (see Section 6.3) in this respect was crucial, as it allowed to get insight on the issue of optimally setting these parameters, thus representing a relevant, yet novel, contribution in itself.

7.2 Open Issues and Further Research

There are some issues that are worth further investigations in the approaches we proposed:

- *Automatically inducing a descriptor from free text.* RecBoost aims to reconcile free text in a fixed attribute schema. It is interesting to investigate the development of an unsupervised approach for the induction of an attribute descriptor from free text. This would still allow reconciliation, even in the absence of any actual knowledge about the textual information at hand.

- *Analyzing new classification models and automatizing the setting of parameters in RecBoost.* The most salient features of RecBoost were the combination of ontology-based generalization with rule-based classification, and the adoption of progressive classification. In this respect, it is interesting to investigate the adoption of alternative strategies for learning local classification models. This line is also related with the effort for identifying a fully-automated technique for setting the parameters of progressive classification, in terms of required classification stages.

- *Exploiting more informative similarity functions for Data Reconciliation.* The de-duplication approach was proved to be quite effective when the available information is based solely on strings, and Jaccard similarity is adopted to compare the features two tuples exhibit. However, when strings are too small or too different to contain enough informative content, the de-duplication task cannot be properly accomplished by the proposed clustering algorithm. To this purpose, we plan to study the extension of the proposed approach to different scenarios, where more informative similarity functions can be exploited. An example is the adoption of the *link-based similarity*. Recently, there has been significant interest in the use of links for improving the reconciliation of data [59]. The central idea is to consider, in addition to the attributes of the tuples to be compared, other possible references to which these are potentially linked. The links may be, for example, co-author links between author references in bibliographic data, hierarchical links between spatial references in geo-spatial data, or co-occurrence links between name references in natural language documents. In a recent approach, Kalashnikov et al. [82] enhanced feature-based similarity between an ambiguous attribute value and the many entity choices available for it, with linkage analysis between the entities (such as affiliation and co-authorship). It is worth noticing that this approach only ensures that references (i.e., "foreign keys") in a database point to the correct entities or analogously, that a feature of a record in a relation exactly matches a feature of a record in a different relation. Moreover, although the proposed technique has been proved effective, it still suffers from the incrementality issues which, instead, are the focus of our data reconciliation technique.

- *Optimizing the setting of parameters in the hierarchical hashing approach.* As previously discussed in this thesis, our refined hashing approach depends on some parameters: the number of min-wise functions at the first and second level of encoding, and the size of q-grams. In the experimental section of Chapter 6, we deeply analyzed how setting these parameters. However, the way to optimally tune them is still an open and interesting problem to further investigate.

References

1. E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *Proc. 10th Int. Conf. on Knowledge Discovery and Data Mining (KDD'04)*, pages 20–29, 2004.
2. E. Agichtein and L. Gravano. SNOWBALL: extracting relations from large plain-text collections. In *Proc. 5th Int. Conf. on Digital Libraries (DL'00)*, pages 85–94, 2000.
3. R. Agrawal and R. Srikant. Searching with numbers. In *Proc. 11th Int. Conf. World Wide Web (WWW11)*, pages 420–431, 2002.
4. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. First edition, 1993.
5. R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proc. 28th Int. Conf. on Very Large Databases (VLDB'02)*, pages 586–597, 2002.
6. D. E. Appelt, J. R. Hobbs, J. Bear, D. J. Israel, and M. Tyson. FASTUS: A finite-state processor for information extraction from real-world text. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI'93)*, pages 1172–1178, 1993.
7. N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
8. L. Baum. An inequality and associated maximization techniques in statistical estimation of probabilistic functions of markov processes. *Inequalities*, (3):1–8, 1972.

9. R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *Proc. KDD'03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27, 2003.
10. A. L. Berger, S. Della Pietra, and V. J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.
11. I. Bhattacharya and L. Getoor. Latent Dirichlet allocation model for entity resolution. Technical Report CS-TR-4740, Computer Science Department, University of Maryland, 2005.
12. M. Bilenko, R. J. Mooney, W. W. Cohen, P. Ravikumar, and S. E. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.
13. D. Bitton and D. J. DeWitt. Duplicate record elimination in large data files. *ACM Transactions on Database Systems*, 8(2):255–265, 1983.
14. A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. 11th Int. Conf. on Computation learning theory (COLT'98)*, pages 92–100, 1998.
15. V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'01)*, pages 175–186, 2001.
16. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. 1984.
17. E. Brill. Some advances in transformation-based part of speech tagging. In *Proc. 12th Nat. Conf. on Artificial Intelligence (AAAI'94)*, pages 722–727, 1994.
18. S. Brin. Extracting patterns and relations from the World Wide Web. In *Proc. Int. Workshop on the Web and Databases (WebDB'98)*, pages 172–183, 1998.
19. A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-Wise independent permutations. In *Proc. 30th Symp. on Theory of Computing (STOC'98)*, pages 327–336, 1998.
20. Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

21. R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.
22. M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proc. 16th Nat. Conf. on Artificial Intelligence (AAAI'99)*, pages 328–334, 1999.
23. A. Chatterjee and A. Segev. Data manipulation in heterogeneous databases. *SIGMOD Record*, 20(4):64–68, 1991.
24. S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'03)*, pages 313–324, 2003.
25. S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *Proc. 21st Int. Conf. on Data Engineering (ICDE'05)*, pages 865–876, 2005.
26. E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
27. P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. 1996.
28. S. Chen and R. Rosenfeld. A survey of smoothing techniques for ME models. *IEEE Transactions on Speech and Audio Processing*, 8(1):37–50, 2000.
29. H. L. Chieu and H. T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *Proc. 18th Nat. Conf. on Artificial Intelligence (AAAI'02)*, pages 786–791, 2002.
30. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. 23rd Int. Conf. on Very Large Data Bases (VLDB'97)*, pages 426–435, 1997.
31. M. Cochinwala, S. Dalal, A.K. Elmagarmid, and V.S. Verykios. Record matching: Past, present and future, 2001.
32. M. Cochinwala, V. Kurien, G. Lalk, and D. Shasha. Improving generalization with active learning. *Information Science*, 137(1-4):1–15, 2001.

33. W. W. Cohen. Integration of heterogeneus databases without common domains using queries based on textual similarity. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'98)*, pages 201–212, 1998.
34. W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, 2000.
35. W. W. Cohen, H. Kautz, and D. McAllester. Hardening soft information sources. In *Proc. 6th Int. Conf. on Knowledge Discovery and Data Mining (KDD'00)*, pages 255–259, 2000.
36. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proc. IJCAI'03 Workshop on Information Integration on the Web*, pages 73–78, 2003.
37. W. W. Cohen and J. Richman. Learning to match and cluster large high dimensional data sets for data integration. In *Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*, pages 475–480, 2002.
38. D. A. Cohn, L. Atlas, and R. E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.
39. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithm.* 1990.
40. J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480, 1972.
41. T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure or how to build a data quality browser. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'02)*, pages 240–251, 2002.
42. D. S. Day, J. S. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain. Mixed-initiative development of language processing systems. In *Proc. 5th Int. Conf. on Applied Natural Language Processing*, pages 348–355, 1997.
43. S. Della Pietra, V. J. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.
44. A. P. Dempster, N. McKenzie Laird, and D. B. Rubin. Maximum likelihood from incomplete data via EM algorithm. *Journal of the Royal Statistical Society*, B(39):1–38, 1977.

45. D. J. DeWitt, J. F. Naughton, and D. A. Schneider. An evaluation of non-equijoin algorithms. In *Proc. 17th Int. Conf. on Very Large Databases (VLDB'91)*, pages 443–452, 1991.
46. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. 1973.
47. M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. TAILOR: A record linkage tool box. In *Proc. 18th Int. Conf. on Data Engineering (ICDE'02)*, pages 17–28, 2002.
48. M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.
49. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.
50. I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
51. S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.
52. D. Freitag. Information extraction from HTML: Application of a general machine learning approach. In *Proc. 15th Nat. Conf. on Artificial Intelligence (AAAI'98)*, pages 517–523, 1998.
53. D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *Proc. AAAI'99 Workshop on Machine Learning for Information Extraction*, 1999.
54. D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proc. 17th Nat. Conf. on Artificial Intelligence (AAAI'00)*, pages 584–589, 2000.
55. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, 1999.
56. R. Gaizauskas and A. Robertson. Coupling information retrieval and information extraction: A new text technology for gathering information from the web.

In *Proc. 5th Conf. on Computer-Assisted Information Searching on Internet (RIAO'97)*, pages 356–370.

57. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. A. Saita. Declarative data cleaning: Language, model, and algorithms. In *Proc. 27th Int. Conf. on Very Large Databases (VLDB'01)*, pages 371–380, 2001.
58. V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French. Clustering large datasets in arbitrary metric spaces. In *Proc. 15th Int. Conf. on Data Engineering (ICDE'99)*, pages 502–511, 1999.
59. L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explorations*, 7(2):3–12, 2005.
60. L .E. Gill. Ox-link: The Oxford medical record linkage system. In *Proc. Int. Workshop on International Record Linkage and Exposition*, pages 15–33, 1997.
61. A. Gionis, D. Gunopulos, and N. Koudas. Efficient and tunable similar set retrieval. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'01)*, pages 247–258, 2001.
62. A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. 25th Int. Conf. on Very Large Data Bases (VLDB'99)*, pages 518–529, 1999.
63. J. Goodman. Exponential priors for maximum entropy models. In *Proc. Conf. on Human Language Technology (NAACL'04)*, pages 305–312, 2004.
64. L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, L. Pietarinen, and D. Srivastava. Using q-grams in a DBMS for approximate string processing. *IEEE Data Engineering Bulletin*, 24(4):28–34, 2001.
65. L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. 27th Int. Conf. on Very Large Databases (VLDB'01)*, pages 491–500, 2001.
66. L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *Proc. 12th Int. Conf. World Wide Web (WWW12)*, pages 90–101, 2003.
67. L. Gu, R. A. Baxter, D. Vickers, and C. Rainsfold. Record linkage: Current practice and future directions. Technical Report 03/83, CSIRO Mathematical and Information Sciences, 2003.

68. S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'98)*, pages 73–84, 1998.
69. S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.
70. S. Gupta, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *Proc. 30th Int. Conf. on Very Large Databases (VLDB'04)*, pages 636–647, 2004.
71. D. Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.
72. T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. 2001.
73. M. A. Hernández and S. J. Stolfo. The merge/purge for large databases. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'95)*, pages 127–138, 1995.
74. M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
75. G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, 2003.
76. R. V. Hogg and A. T. Craig. *Introduction to Mathematical Statistics*. Fourth edition, 1978.
77. P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. 30th Symp. on Theory of Computing (STOC'98)*, pages 604–613, 1998.
78. M. A. Jaro. Unimatch: A record linkage system: User's manual. Technical report, U.S. Bureau of the Census, Washington, D.C., 1976.
79. M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.
80. T. Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines*.

81. R. Jones, A. McCallum, K. Nigam, and E. Riloff. Bootstrapping for text learning tasks. In *Proc. IJCAI'99 Workshop on Text Mining: Foundations, Techniques and Applications*, 1999.

82. D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *Proc. 5th Int. Conf. on Data Mining (SIAM'05)*, pages 262–273, 2005.

83. J. Kim and D. I. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):713–724, 1995.

84. D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proc. 14th Int. Conf. on Machine Learning (ICML'97)*, pages 170–178, 1997.

85. N. Koudas, A. Marathe, and D. Srivastava. Flexible string matching against large databases in practice. In *Proc. 30th Int. Conf. on Very Large Databases (VLDB'04)*, pages 1078–1086, 2004.

86. K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

87. J. Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6:225–242, 1992.

88. J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th Int. Conf. on Machine Learning (ICML'01)*, pages 282–289, 2001.

89. G. M. Landau and U. Vishkin. Fast parallel and serial string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

90. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversal. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

91. E. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *Proc. 9th Int. Conf. on Data Engineering (ICDE'93)*, pages 294–301, 1993.

92. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, 1998.

93. R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proc. 6th Int. Conf. on Natural Language Learning (CoNLL'02)*, pages 49–55, 2002.
94. N. Mamoulis. Efficient processing of joins on set-valued attributes. In *Proc. ACM Int. Conf. on Menagement of Data (SIGMOD'03)*, pages 157–168, 2003.
95. A. McCallum. MALLET: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.
96. A. McCallum, D. Freitag, and F. C. N. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proc. 17th Int. Conf. on Machine Learning (ICML'00)*, pages 591–598, 2000.
97. A. McCallum and D. Jensen. A note on the unification of information extraction and data mining using conditional probability, relational models. In *Proc. IJCAI'03 Workshop on Learning Statistical Models from Relational Data*, 2003.
98. A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. 6th Int. Conf. on Knowledge Discovery and Data Mining (KDD'00)*, pages 169–178, 2000.
99. A. McCallum and B. Wellner. Conditional models of identity uncertainty with application to noun coreference. In *Proc. 18th Int. Conf. on Neural Information Processing Systems (NIPS'04)*, 2004.
100. X. Meng and D. B. Rubin. Maximum likelihhod via the ECM algorithm: A general framework. *Biometrika*, 80:267–278, 1993.
101. G. A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.
102. R. Mitkov. *Anaphora Resolution*. First edition, 2002.
103. A. E. Monge and C. P. Elkan. The field matching problem: Algorithms and applications. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 267–270, 1996.
104. A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. 2nd SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'97)*, pages 23–29, 1997.

105. G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

106. S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

107. H. B. Newcombe. *Handbook of Record Linkage*. Oxford University Press, 1988.

108. H. B. Newcombe and J. M. Kennedy. Record linkage: Making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, 1962.

109. H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

110. H. Pasula, B. Marthi, B. Milch, S. Russel, and I. Shpister. Identity uncertainty and citation matching. In *Proc. 16th Int. Conf. on Neural Information Processing Systems (NIPS'02)*, 2002.

111. M. Perkowitz, R. B. Doorenbos, O. Etzioni, and D. S. Weld. Learning to understand information on the Internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, 1997.

112. L. Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12):39–44, 1990.

113. L. Philips. The double metaphone search algorithm. *C/C++ Users Journal*, 18(5), 2000.

114. J. C. Pinheiro and D. X. Sun. Methods for linking and mining heterogeneous databases. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98)*, pages 309–313, 1998.

115. D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proc. 26th Int. Conf. on Research and Development in Information Retrieval (SIGIR'03)*, pages 235–242, 2003.

116. L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 1989.

117. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2003.

118. P. Ravikumar and W. W. Cohen. A hierarchical graphical model for record linkage. In *Proc. 20th Int. Conf. on Uncertainty in Artificial Intelligence (UAI'04)*, 2004.
119. E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proc. 11th Nat. Conf. on Artificial Intelligence (AAAI'93)*, pages 811–816, 1993.
120. E. Riloff. Automatically generating extraction patterns from untagged text. In *Proc. 13th Nat. Conf. on Artificial Intelligence (AAAI'96)*, pages 1044–1049, 1996.
121. E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proc. 16th Nat. Conf. on Artificial Intelligence (AAAI'99)*, pages 474–479, 1999.
122. E. S. Ristad and P. N. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998.
123. E. A. Rundensteiner. Letter from the special issue editor. *IEEE Data Engineering Bulletin*, 22(1):2, 1999.
124. S. Sarawagi. Letter from the special issue editor. *IEEE Data Engineering Bulletin*, 23(4):2, 2000.
125. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDDD'02)*, pages 269–278, 2002.
126. S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proc. ACM Int. Conf. on Menagement of Data (SIGMOD'04)*, pages 743–754, 2004.
127. K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *Proc. AAAI'99 Workshop on Machine Learning for Information Extraction*, pages 37–42, 1999.
128. F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. Conf. on Human Language Technology (NAACL'03)*, pages 134–141, 2003.
129. P. Singla and P. Domingos. Multi-relational record linkage. In *Proc. 3rd Workshop on Multi-Relational Data Mining (MRDM'04)*, 2004.
130. M. Skounakis, M. Craven, and S. Ray. Hierarchical hidden Markov models for information extraction. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI'03)*, pages 427–433, 2003.

131. D. D. Sleator and D. Temperley. Parsing English with a link grammar. In *Proc. 3rd Int. Workshop on Parsing Technologies (IWPT'93)*, 1993.

132. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

133. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

134. S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. CRYSTAL: Inducing a conceptual dictionary. In *Proc. 14th Int. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pages 1314–1319, 1995.

135. A. Soffer, D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, and Y. S. Maarek. Static index pruning for information retrieval systems. In *Proc. 24th Int. Conf. on Research and Development in Information Retrieval (SIGIR'01)*, pages 43–50, 2001.

136. R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 21th Int. Conf. on Very Large Databases (VLDB'95)*, pages 407–419, 1995.

137. A. Stolcke. *Bayesian Learning of Probabilistic Language Models.* PhD thesis, University of California, Berkeley, CA, 1994.

138. E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In *Proc. 3rd Annual European Symposium (ESA'98)*, pages 327–340, 1995.

139. R. L. Taft. Name search techniques. Technical Report 1, New York State Identification and Intelligent System, Albany, NY, 1970.

140. R. E. Tarjan. Efficiency of a good but non linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

141. S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information System*, 26(8):607–633, 2001.

142. S. Tejada, C. A. Knoblock, and S. Minton. Learning domain independent string transformation weights for high accuracy object identification. In *Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*, pages 350–359, 2002.

143. B. Tepping. A model for optimum linkage of records. *Journal of the American Statistical Association*, 63(324):1321–1332, 1968.

144. D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical Analysis of Finite Mixture Distributions*. 1988.

145. E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

146. J. R. Ullmann. A binary n-gram technique for automatic correction of substitution, deletion, insetion and reversal errors in words. *The Computer Journal*, 20(2):141–147, 1977.

147. V. S. Verykios, A. K. Elmagarmid, and E. N. Houstis. Automating the approximate record matching process. *Information Sciences*, 126(1-4):83–98, 2000.

148. A. J. Viterbi. Error bounds for convolutional codes and an asymtotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267, 1967.

149. Y. R. Wang and S. E. Madnick. The inter-database instance identification problem in integrating autonomous system. In *Proc. 5th Int. Conf. on Data Engineering (ICDE'89)*, pages 46–55, 1989.

150. R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. on Very Large Data Bases (VLDB'98)*, pages 194–205, 1998.

151. J. Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

152. J. Widom. Research problems in data warehousing. In *Proc. 4th Int. Conf. on Information and Knowledge Management (CIKM'95)*, pages 25–30, 1995.

153. W. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, Wachington D.C., 1999.

154. W. E. Winkler and Y. Thibaudeau. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. Technical report, Statistical Research Report Series RR91/09, U.S. Bureau of the Census, Washington, D.C., 1991.

155. W. E. Yancey. Bigmatch: A program for extracting probable matches from a large file for record linkage. Technical Report Statistical Research Report Series RRS2002/01, U.S. Bureau of the Census, Washington D.C., 2002.

156. W. E. Yancey. Evaluating string comparator performance for record linkage. Technical Report Statistical Research Report Series RRS2005/05, U.S. Bureau of the Census, Washington D.C., 2005.
157. R. Yangarber. *Scenario Customization of Information Extraction.* PhD thesis, Courant Institute of Mathematical Sciences. New York University, 2000.
158. R. Yangarber. Counter-training in discovery of semantic patterns. In *Proc. 41th Annual Meeting of the Association for Computational Linguistics (ACL'03)*, pages 343–350, 2003.
159. R. Yangarber and R. Grishman. NYU: Description of the Proteus/PET system as used for MUC-7 ST. In *Proc. 7th Message Understanding Conference (MUC-7)*, 1998.
160. J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature file for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, 1998.