UNIVERSITÀ DELLA CALABRIA

**Dipartimento di Elettronica,
Informatica e Sistemistica**

**Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica
XXI Ciclo**

Tesi di Dottorato

Modelling Complex Data Mining Applications in a Formal Framework

Antonio Locane

UNIVERSITÀ DELLA CALABRIA

**Dipartimento di Elettronica,
Informatica e Sistemistica**

**Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica
XXI Ciclo**

Tesi di Dottorato

Modelling Complex Data Mining Applications in a Formal Framework

Antonio Locane

Coordinatore
Prof. Domenico Talia

Supervisori
Prof. Domenico Saccà

Dott. Giuseppe Manco

DEIS- DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA
Novembre 2008

Settore Scientifico Disciplinare: ING-INF/05

Dedicated to Sara

Contents

Part I

Introduction

1

Introduction

1.1 Motivation

The fast growth of electronic data management methods in the last years, has lead some to call recent times as the *Information Age*. Powerful systems for collecting and managing data are currently exploited in virtually all large and mid-range companies, it is very difficult to find a *transaction* that does not generate a computer record somewhere.

More and more operations are being automated and computerized, each of them generates data, and all these data hold valuable information (i.e. trends and patterns) which could in principle be used to improve business decisions and optimize success. However, today's databases contain so much data that it is practically impossible to manually analyze them for valuable decision-making information. In many cases, hundreds of independent attributes need to be simultaneously considered in order to accurately model a system behavior, therefore, humans need assistance in their analysis capacity.

This need for automated extraction of useful knowledge from huge amounts of data is widely recognized, and leads to a rapidly developing market of automated analysis tools to discover strategic information hidden in very large databases. The main needed capability is to analyze raw data and present the extracted high level information to the analyst or decision-maker.

Knowledge discovery in databases (KDD or simply KD) addresses the problem mentioned above by developing automatic and intelligent techniques for automated discovery of useful and interesting patterns (knowledge) in raw data. The main effort in the knowledge discovery community has so far been devoted to the development of efficient mining algorithms, but there are not many contributions aimed at tackling and ultimately solving the whole problem. In other words, there is no set of widely accepted techniques and methodologies to support the entire knowledge discovery process: even if many knowledge discovery systems exist, each of them use its own methodology.

The need for a systematic description of the knowledge discovery process has been recognized in the KD community. However, KD is a complex inter-

disciplinary task, both data-centered and human-centered. So, if on one hand it is naturally desirable to have an unifying platform (hopefully built on formal basics) to perform the overall process, on the other hand the aforementioned complexity makes the development of such a platform a great challenge.

1.2 Contribution of the thesis

This thesis tries to identify some common high-level problems that arise when attempting to define a framework for the knowledge discovery process. Such problems, as well as general open issues, are examined from a theoretical and practical point of view, and then a possible solution is outlined. A discussion is also made on basic requirements a KD framework must have, in order to be sufficiently expressive to cope with usage patterns that commonly arise in KD processes. As knowledge discovery is a wide, open and evolving topic, a systematic approach aimed at modeling such process has to be open and extensible.

To this purpose, we propose a framework that supports an operations flow in which mining tasks can be performed, and the resulting models can be further processed, visualized, evaluated and finally activated to leverage the overall discovery process. In our vision, the essence of a knowledge discovery process is the interaction between two apparently separate worlds: the *data world* and the *model world*. Our proposal is based on an algebraic framework, referred to as 2W Model, that defines the knowledge discovery process as a progressive combination of mining and querying operators.

After analyzing typical problems related with the definition of a KD framework, a possible implementation of such framework is proposed, which attempts to solve all aforementioned issues. The proposed implementation is compliant with the theoretic view defined in 2W Model, and also introduces new features in order to deal with practical real-world problems such as managing complex data types and efficiently storing and accessing huge amounts of data. In addition, particular attention is given to data and models exploration, defining suitable environments where the analyst can effectively and easily examine and inspect both data and models, taking advantage of descriptive statistics, explanatory charts, and several models visualizations metaphors.

In order to demonstrate the defined framework capabilities and better describe the framework itself, an applicative example is presented as a *proof of the concept*. In particulary, a novel approach for reconciling tuples stored as free text into an existing attribute schema is proposed. The basic idea is to subject text to *progressive classification*, that is a multi-stage classification scheme where, at each intermediate stage, a classifier is learnt that analyzes the textual fragments not reconciled at the end of previous steps. Classification is accomplished by an ad-hoc exploitation of traditional association mining algorithms, and is supported by a data transformation scheme which

takes advantage of domain-specific dictionaries and ontologies. A key feature is the capability of progressively enriching the available ontology with results coming up from previous classification stages, thus significantly improving the overall classification accuracy. An extensive experimental evaluation shows the effectiveness of such approach.

1.3 Organization

The rest of the thesis is organized as follows.

Chapter 2 closes part I, giving some basic definitions regarding both the knowledge discovery process and the central data mining phase.

In part II, a formal knowledge discovery framework is proposed. First of all, chapter 3 introduces known design issues and main challenges, as well as basic requirements the framework must fulfill. The 2W Model is then presented in chapter 4, and finally a possible effective implementation of the proposed theoretic framework is given in chapter 5.

In part III, an applicative example is used to better explain the proposed framework capabilities. In chapter 6, an innovative methodology for reconciling free text into a given attribute schema is proposed. Chapter 7 describes results coming from extensive experimental evaluation sessions, and also presents a real-world case study.

Part IV is essentially dedicated to draw some conclusions regarding both the KD framework definition/implementation defined in part II, and the methodology proposed in part III.

2

Basic definitions

2.1 The knowledge discovery process

Historically the notion of finding useful patterns in data has been given a variety of names, such as data mining, knowledge extraction, information discovery, information harvesting, data archaeology, data pattern processing, and much others. The term data mining has been mostly used by statisticians, data analysts, and the management information systems communities, and has also gained popularity in the database field.

The term KDD was coined at the first KDD workshop in 1989 (see [79]) to emphasize that *knowledge* is the end product of a datadriven discovery, and since then it has been popularized in artificial intelligence and machine learning.

Knowledge Discovery in Databases (i.e. the process of finding *nuggets* of knowledge in data) is a rapidly growing field, whose development is driven by strong research interests as well as urgent practical, social, and economical needs. The KDD process is a complex task, heavily dependent on the problem and on the data at hand, and covers a wide range of applicative domains (e.g. retail, marketing, finance, e-commerce, biology, privacy), several models of representing extracted patterns and rules (e.g. classification models, association rules, sequential patterns, clusters) and a large number of algorithms for data preprocessing, model extraction and model reasoning.

The knowledge discovery goals are defined by the intended use of the process. Two types of goals can be distinguished:

1. Verification, where the goal is limited to verifying the user's hypothesis;
2. Discovery, where the goal is to autonomously finds new patterns.

The Discovery goal can further be divided into:

- Prediction, where the goal is to find patterns for the purpose of predicting the future behavior of some entities;

- Description, where the goal is to find patterns for the purpose of presenting them to a user in a humanunderstandable form.

Although the boundaries between prediction and description are not sharp (some of the predictive models can be descriptive, to the degree that they are understandable, and vice versa), the distinction is useful for understanding the overall discovery goal. The relative importance of prediction and description for particular data mining applications can vary considerably, for example, prediction may be used to validate a discovered hypothesis.

There is still some confusion about the terms Knowledge Discovery in Databases (KDD) and data mining (DM) and often these two terms are used interchangeably. Actually, KDD refers to the overall process of discovering useful knowledge from data while data mining refers to a particular step in this process, in particular, data mining is the application of specific algorithms for extracting patterns from data.

Hereafter, the term KDD will be used to denote the overall process of turning low-level data into high-level knowledge. KDD can be defined as the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

As described in the CRISP-DM process model [17] (see Fig. 2.1), KDD typically consists of several repeated phases, including: business problem understanding, data understanding, data preparation, incorporating appropriate prior knowledge, modeling (or data mining), proper interpretation of the results, evaluation and deployment. The development of KDD solutions requires then to specify the tasks at each phase and the interactions/dependencies among them. Most of the times, this results in a complex process, requiring to combine different sources of data and knowledge, and with many tasks iterated in order to ensure that useful knowledge is derived from the data. Blind application of data mining methods can be a dangerous activity easily leading to discovery of meaningless patterns.

KDD has evolved, and continues evolving, from the intersection of research fields such as machine learning, pattern recognition, databases, statistics, artificial intelligence, knowledge acquisition for expert systems, data visualization, and high performance computing. KDD overlaps with machine learning and pattern recognition in the study of particular data mining theories and algorithms: means for modeling data and extracting patterns. KDD also has much in common with statistics, particularly exploratory data analysis methods. The statistical approach offers precise methods for quantifying the inherent uncertainty which results when one tries to infer general patterns from a particular sample of an overall population. KDD software systems often embed particular statistical procedures for sampling and modeling data, evaluating hypotheses, and handling noise.

In addition to its strong relation to the database field (the 2nd 'D' in KDD), another related area is data warehousing, which refers to the popular business trend for collecting and cleaning transactional data to make them

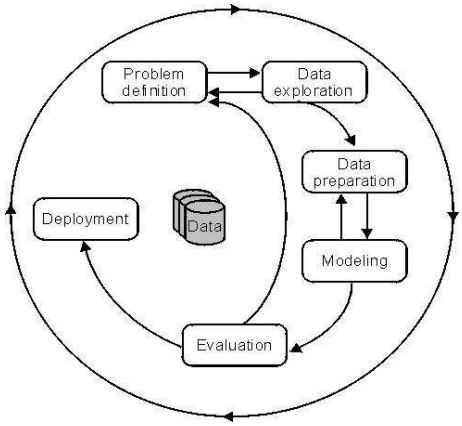


Fig. 2.1. The CRISP-DM process model

available for online analysis and decision support. A popular approach for analysis of data warehouses has been called OLAP (online an alytical processing). OLAP tools focus on providing multidimensional data analysis, which is superior to SQL in computing summaries and breakdowns along many dimensions. OLAP tools are targeted towards simplifying and supporting interactive data analysis, while the KDD tool's goal is to automate as much of the process as possible.

Knowledge Discovery in Databases can be formally defined as (see [37])

> *the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.*

In the definition above, *data* is a set of facts (e.g., cases in a database) and *pattern* is an expression in some language describing a subset of the data or a model applicable to that subset. Hence extracting a pattern also designates fitting a model to data, finding structure from data, or in general any highlevel description of a set of data. The term *process* implies that KDD is comprised of many steps, and requires repeated multiple iterations. *Nontrivial* means that some search or inference is involved (i.e. it is not a straightforward computation of predefined quantities like computing the average value of a set of numbers). The discovered patterns should be *valid* on new data with some degree of certainty. We also want patterns to be *novel* (at least to the system, and preferably to the user) and potentially *useful*, (i.e., lead to some benefit to the user/task). Finally, the patterns should be *understandable*, if not immediately then after some postprocessing.

The above implies that some quantitative measures for evaluating extracted patterns have to be defined. In many cases, it is possible to define measures of certainty (e.g., estimated prediction accuracy on new data) or utility (e.g. gain, perhaps in dollars saved due to better pre dictions or speedup

in response time of a system). On the other hand, notions such as novelty and understandability are much more subjective, whereas in certain contexts understandability can be estimated by simplicity (e.g., the number of bits to describe a pattern). An important notion, called interestingness, is usually taken as an overall measure of pattern value, combining validity, novelty, usefulness, and simplicity. Interestingness functions can be explicitly defined or can be manifested implicitly via an ordering placed by the KDD system on the discovered patterns or models.

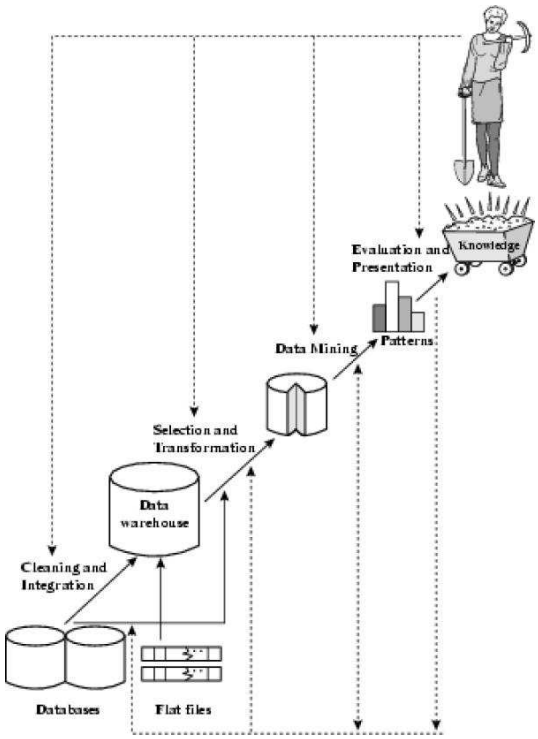


Fig. 2.2. The KDD process

As can be seen in figure 2.2, the KDD process can involve significant iteration and may contain loops between any two steps.

The aforementioned figure depicts the basic flow of steps, although not the potential multitude of iterations and loops. Most previous work on KDD has focused on the mining step, however, in real world application there are several other preliminary and posterior operations which play a very important role, those operations concern:

- Developing an understanding of the application domain;

- Identifying the goals of the process from the customer's viewpoint;
- Acquiring or selecting a target data set, or focusing on a subset of variables or data samples;
- Integrating and checking the data set;
- Performing data cleaning, preprocessing, and transformation (e.g. removing noise, deciding strategies for handling missing data fields, etc...);
- Reducing and projecting data (e.g. using dimensionality reduction or transformation methods to reduce the effective number of variables under consideration, or finding invariant representations for the data;
- Integrating data with pre-existent knowledge;
- Matching the goals of the KDD process to a particular data mining method (e.g., summarization, classification, regression, clustering, etc...);
- Choosing suitable algorithms to be used for searching for patterns in the data, and deciding which parameters may be appropriate;
- Matching a particular data mining method with overall criteria (e.g., the enduser may be more interested in understanding the model than its predictive capabilities);
- Performing the mining step, searching for patterns of interest in the data;
- Interpreting the extracted patterns/models visualizing them in a human-readable form, or visualizing the data given the extracted models, possibly returning to any of the preceding steps;
- Results testing and verification;
- Consolidating discovered knowledge documenting it and reporting it to interested parties, as well as checking for and resolving potential conflicts with previously believed or extracted knowledge;
- Using mined models (e.g. incorporating them into other systems for further action).

In real world application, domain knowledge usually resides in an expert's brain, so it is natural that such an expert would better guide the process, especially during the initial data preprocessing and during the final knowledge identification. Furthermore, it would be convenient to be able to use an existing domain knowledge stored in a knowledge base, to integrate and possibly refine the newly discovered knowledge coming as the outcome of the KDD process.

Exploratory Data Analysis (EDA) plays also an important role in the KDD process. Such analysis is the simply interactive exploration of a data set without heavy dependence on preconceived assumptions and models, thus attempting to identify interesting patterns. Graphic representations of the data are used very often to exploit the power of the eye and human intuition. While there are dozens of software packets available that were developed exclusively to support data exploration, it might also be desirable to integrate these approaches into an overall KDD environment. The human eye-brain system itself still remains the best pattern-recognition device known. Visual-

ization techniques may range from simple scatter plots and histogram plots over parallel coordinates to 3D movies.

2.2 The data mining step

Data mining is just one step in the overall KDD process, consisting of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data. It have to be noticed that the space of patterns is often infinite, and the enumeration of patterns involves some form of search in this space. As a consequence, practical computational constraints place severe limits on the subspace that can be explored by a data mining algorithm. It should also be noted that several methods with different goals may be applied successively to achieve a desired result. For example, to determine which customers are likely to buy a new product, a business analyst might need to first use clustering to segment the customer database, then apply regression to predict buying behavior for each cluster.

The term *Data Mining* (DM) will be used to address the extraction of patterns or models from observed data. Although at the core of the knowledge discovery process, the mining step usually takes only a small part (15% to 25%) of the overall effort.

The data mining component of the KDD process is concerned with the algorithmic means by which patterns are extracted and enumerated from data. The overall KDD process (see fig. 2.2) includes the evaluation and possible interpretation of the *mined* patterns to determine which patterns may be considered new *knowledge*.

The objective of this section is to present a very brief overview of the primary goals of data mining, along with a description of the main methods used to address these goals, and a very brief overview of data mining algorithms which incorporate these methods.

Most data mining methods are based on tried and tested techniques from machine learning, pattern recognition, and statistics: classification, clustering, regression, and so forth. The set of different algorithms under each of these headings can often be quite bewildering to both the novice and experienced data analyst. However, it should be emphasized that of the very many data mining methods advertised in the literature, there are really only a few fundamental techniques.

The goals of prediction and description are achieved via the following primary data mining methods.

Classification

Learning a function that maps (classifies) a data item into one of several predefined classes. Given a set of predefined categorical classes, determine to

which of these classes a specific data item belongs. For example, given classes of patients that correspond to medical treatment responses, identify the form of treatment to which a new patient is most likely to respond.

Regression

Learning a function which maps a data item to a realvalued prediction variable and discovering functional relationships between variables. Given a set of data items, regression is the analysis of the dependency of some attribute values upon the values of other attributes in the same item, and the automatic production of a model that can predict these attribute values for new records. For example, given a data set of credit card transactions, build a model that can predict the likelihood of fraudulence for new transactions.

Clustering

Identifying a finite set of categories or clusters to describe the data. Given a set of data items, partition this set into a set of classes such that items with similar characteristics are grouped together. Clustering is best used for finding groups of items that are similar. For example, given a data set of customers, identify subgroups of customers that have a similar buying behavior. Closely related to clustering is the method of probability density estimation which consists of techniques for estimating from data the joint multivariate probability density function of all of the variables/fields in the database.

Dependency Modeling (or Association Analysis, or Link Analysis)

Finding a model which describes significant dependencies between variables. Given a set of data items, identify relationships between attributes and items such as the presence of one pattern implies the presence of another pattern. These relations may be associations between attributes within the same data item (e.g. "Out of the shoppers who bought milk, 64% also purchased bread") or associations between different data items (e.g. "Every time a certain stock drops 5%, a certain other stock raises 13% between 2 and 6 weeks later"). The investigation of relationships between items over a period of time is also often referred to as *sequential pattern analysis*.

Outlier Detection (or Anomaly Detection)

An outlier is an observation that lies at an abnormal distance from other values in a random sample from a population. In a sense, this definition leaves it up to the analyst (or a consensus process) to decide what will be considered abnormal. Before abnormal observations can be singled out, it is necessary to characterize normal observations.

Having outlined the general methods of data mining, the next step is to construct specific algorithms to implement these methods. One can identify three primary components in any data mining algorithm: model representation, model evaluation, and search. This reductionist view is not necessarily complete or fully encompassing: rather, it is a convenient way to express the key concepts of data mining algorithms in a relatively unified and compact manner.

Model representation can be viewed as the language used to describe discoverable patterns. If the representation is too limited, then no amount of training time or examples will produce an accurate model for the data. It is important that a data analyst fully comprehend the representational assumptions which may be inherent in a particular method. It is equally important that an algorithm designer clearly state which representational assumptions are being made by a particular algorithm. Note that more powerful representational power for models increases the danger of over-fitting the training data resulting in reduced prediction accuracy on unseen data.

Model evaluation criteria are quantitative statements of how well a particular pattern (a model and its parameters) meet the goals of the KDD process. For example, predictive models are often judged by the empirical prediction accuracy on some test set. Descriptive models can be evaluated along the dimensions of predictive accuracy, novelty, utility, and understandability of the fitted model.

Search method consists of two components: parameter search and model search. Once the model representation and the model evaluation criteria are fixed, then the data mining problem has been reduced to purely an optimization task: find the parameters/models selected from the set of parameters/models that can be defined given the chosen representation, which optimize the evaluation criteria. In parameter search the algorithm must search for the parameters which optimize the model evaluation criteria given observed data and a fixed model representation. Model search occurs as a loop over the parameter search method: the model representation is changed so that families of models are considered.

It should be clear from the above that data mining is not a single technique, any method that will help to get more information out of data is useful. Different methods serve different purposes, each method offering its own advantages and disadvantages. Most methods commonly used for data mining can be classified into the following groups.

Statistical Methods

Historically, statistical work has mainly focused on testing preconceived hypotheses and on fitting models to data. Statistical approaches usually rely on an explicit underlying probability model. In addition, it is generally assumed that these methods will be used by statisticians, and hence human intervention is required for the generation of candidate hypotheses and models.

Case-Based Reasoning

Case-based reasoning is a technology that tries to solve a given problem by making direct use of past experiences and solutions. A case is usually a specific problem that has been previously encountered and solved. Given a particular new problem, case-based reasoning examines the set of stored cases and finds similar ones. If similar cases exist, their solution is applied to the new problem, and the problem is added to the case base for future reference.

Neural Networks

Neural networks are a class of systems modeled after the human brain. As the human brain consists of millions of neurons that are interconnected by synapses, neural networks are formed from large numbers of simulated neurons, connected to each other as brain neurons. Like in the human brain, the strength of neuron interconnections may change (or be changed by the learning algorithm) in response to a presented stimulus or an obtained output, which enables the network to learn.

Decision Trees

A decision tree is a tree where each non-terminal node represents a test or decision on the considered data item. Depending on the outcome of the test, a certain branch is chosen. To classify a particular data item, one starts at the root node and follows the assertions down until a terminal node (or leaf) is reached. When a terminal node is reached, a decision is made. Decision trees can also be interpreted as a special form of a rule set, characterized by their hierarchical organization of rules.

Rule Induction

Rules state a statistical correlation between the occurrence of certain attributes in a data item, or between certain data items in a data set. The general form of an association rule is $X_1 \wedge ... \wedge X_n \rightarrow Y[C, S]$, meaning that the attributes $X_1, ..., X_n$ predict Y with a confidence C and a significance S.

Bayesian Belief Networks

Bayesian belief networks are graphical representations of probability distributions, derived from co-occurrence counts in the set of data items. Specifically, a bayesian belief network is a directed, acyclic graph, where the nodes represent attribute variables and the edges represent probabilistic dependencies between the attribute variables. Associated with each node are conditional probability distributions that describe the relationships between the node and its parents.

Genetic algorithms (or Evolutionary Programming)

Genetic algorithms and evolutionary programming are algorithmic optimization strategies that are inspired by the principles observed in natural evolution. Of a collection of potential problem solutions that compete with each other, the best solutions are selected and combined with each other. In doing so, one expects that the overall goodness of the solution set will become better and better, similar to the process of evolution of a population of organisms. Genetic algorithms and evolutionary programming are used in data mining to formulate hypotheses about dependencies between variables, in the form of association rules or some other internal formalism.

Fuzzy Sets

Fuzzy sets form a key methodology for representing and processing uncertainty. Uncertainty arises in many forms in today's databases: imprecision, non-specificity, inconsistency, vagueness, etc. Fuzzy sets exploit uncertainty in an attempt to make system complexity manageable. As such, fuzzy sets constitute a powerful approach to deal not only with incomplete, noisy or imprecise data, but may also be helpful in developing uncertain models of the data that provide smarter and smoother performance than traditional systems. Since fuzzy systems can tolerate uncertainty and can even utilize language-like vagueness to smooth data lags, they may offer robust, noise tolerant models or predictions in situations where precise input is unavailable or too expensive.

Rough Sets

A rough set is defined by a lower and upper bound of a set. Every member of the lower bound is a certain member of the set. Every non-member of the upper bound is a certain non-member of the set. The upper bound of a rough set is the union between the lower bound and the so-called boundary region. A member of the boundary region is possibly (but not certainly) a member of the set. Therefore, rough sets may be viewed as fuzzy sets with a three-valued membership function (yes, no, perhaps). Like fuzzy sets, rough sets are a mathematical concept dealing with uncertainty in data. Also like fuzzy sets, rough sets are seldom used as a stand-alone solution; they are usually combined with other methods such as rule induction, classification, or clustering methods.

An important point is that each technique typically suits some problems better than others. For example, decision tree classifiers can be very useful for finding structure in highdimensional spaces and are also useful in problems with mixed continuous and categorical data (since tree methods do not require distance metrics). However, classification trees may not be suitable for

problems where the true decision boundaries between classes are described, for example, by a 2ndorder polynomial). Thus, there is no *universal* data mining method and choosing a particular algorithm for a particular application is something of an art. In practice, a large portion of the applications effort can go into properly formulating the problem (asking the right question) rather than in optimizing the algorithmic details of a particular data mining method.

Part II

Defining and realizing a knowledge discovery framework

3

Design issues

3.1 Fundamental aspects

The process of analyzing data can be roughly represented as an interaction between the data mining engine and the user, where the user formulates a query describing the patterns of his/her interest and the mining engine returns the patterns by exploiting either domain-specific or physical optimizations. In such a scenario, research on data mining has mainly focused on the definition and implementation of specific data mining engines and mining algorithms.

Each algorithm describes specific operations to be performed in order to build models from data, however, independently from the way models can be built, there are some main components which characterize the overall process:

- the source data to analyze;
- the patterns to discover;
- the criterion for determining the usefulness of patterns;
- the background knowledge.

The first step in a knowledge discovery task is to identify the data to analyze. These data represent the main input which the mining algorithm has to take into account. Let Σ denote the primary knowledge sources. Σ is a set of entities, equipped with a set of characterizing properties, which are of interest for analysis purposes. Identifying relevant entities and properties is a crucial task, since it requires a combination of basic operations which include data integration and selection (i.e., the identification of the data relevant to the analysis task, and its retrieval from a set of possibly heterogenous data sources), and data cleaning and transformation (i.e., the processing of noisy, missing or irrelevant data, and its transformation into appropriate forms). Although the quality of the extracted patterns strongly relies on the effectiveness of such operations, poor attention has been devoted to such a delicate task in the current literature: as a result, the management of knowledge sources is still informal and ad-hoc, and the current data mining tools provide little support

to the related operations. Thus, typically a knowledge expert has the burden to manually study and preprocess the data with several loosely-coupled tools, and to select the most appropriate reorganization of such data that is suitable for the extraction of patterns of interest.

Patterns are sentences about primary knowledge sources, expressing rules, regularities, or models that hold on the entities in Σ. Patterns are characterized by a language $\mathcal{L}$ which is fixed in advance, describing the set of all possible properties which hold in the primary knowledge sources. In this respect, a data mining algorithm describes a way to explore this set of all possible patterns, and to detect the patterns which are of interest according to given criteria. Patterns characterize a data mining task or method and they can be categorized as descriptive (when they describe or summarize the entities within the source data) and predictive (when they characterize a property or a set of properties of an entity according to the values exhibited by other properties).

A search criterion is defined on the involved entities and determines whether a given pattern in the pattern language $\mathcal{L}$ is potentially useful. This construct can be captured by a boolean constraint predicate q, which can be described as relative to some pattern $l \in \mathcal{L}$ and possibly to a set Σ of entities. If $q(l, \Sigma)$ is true, then l is potentially useful. The KDD task then is to find the set $\{l \in \mathcal{L} \mid q(l, \Sigma)\}$ is true. Patterns can be computed as a result of a post-processing operation (among all the patterns discovered thus far, which are the ones satisfying such a condition), or they can be pushed into a specialized mining engine in order to solve them.

Sometimes, the domain knowledge that is already known in advance can be incorporated in each of the previous dimensions. It can be used to enrich the primary knowledge or to derive good initial hypotheses to start the pattern search. Background knowledge can be exploited in the search strategy for pruning the pattern space or it can serve as a reference for interpreting discovered knowledge. Furthermore, since it is difficult to define adequate statistical measures for subjective concepts like novelty, usefulness, and understandability, background knowledge can be helpful in capturing such concepts more accurately.

Clearly, any realistic knowledge discovery process is not linear, but rather iterative and interactive. Any one step may result in changes in earlier steps (see Fig. 2.2), thus producing a variety of feedback loops. As a consequence, the design of final applications is still a *handmade* process, aimed at smoothly composing algorithm libraries, proprietary APIs, SQL queries and stored procedure calls to RDBMS, and still a great deal of ad-hoc code. This motivates the development of tools that support the entire KDD process, rather than just the core data-mining step.

As stated in [54], the current situation is very similar to that in DBMS in the early 1960s, when each application had to be built from scratch, without the benefit of dedicated database primitives provided later by SQL and relational database APIs. Such primitives are sufficient to support the vast majority of business applications for which present DBMS are mainly designed

for. However, there are no similar primitives sufficient to capture the emerging family of new applications dealing with knowledge discovery.

In a way, todays techniques of data mining would more appropriately be described as *file mining* since they assume a loose coupling between the data mining engine providing algorithms and a framework where to design the entire KDD process. Hence, real data mining/analysis applications call for a framework which adequately supports knowledge discovery as a multi-step process, where the input of one (mining) operation can be the output of another.

One may argue that performance improvement of I/O operations alone would have never triggered the DBMS research field nearly 40 years ago. If queries were predefined and their number was limited it would be sufficient to develop highly tuned, stand-alone, library routines [54]. But queries are of course not predefined, so query languages such SQL, along with researches concerning query optimization and transaction processing, were the driving ideas behind the tremendous growth of database field in the last four decades. To be more specific, it was the ad hoc nature of querying that created a challenge to build general-purpose query optimizers.

However, in a KDD environment, query have to be much more general than SQL, and the queried objects have to be far more complex than records (tuples) in relational database. Similarly, KDD query optimization would be more challenging than relational query optimization due to the higher expressive power of KDD queries. Another difficulty is that the border between querying and *discovery* is fuzzy: *discovery* is a very fuzzy term and is often misused to describe the results of standard SQL querying.

The definition of a KDD framework or a an effective language support for the KDD process are still open problems with several different proposals, but without a predominant one.

3.2 Main challenges

Knowledge discovery is not simply machine learning with large data sets, and essential needs are not only concerned with performance of mining operations running on large persistent data sets and involving expensive I/O. Although improving performance is an important issue, it is not sufficient to trigger a qualitative change in system capabilities.

Some of the current primary research and application challenges for KDD are cited in the following list, which is intended to give the reader a feel for the types of problems that KDD practitioners wrestle with:

- massive datasets;
- high dimensionality;
- overfitting and assessing statistical significance;
- missing and noisy data;

- complex relationship between fields;
- understandability of patterns;
- integration with other systems;
- integration with prior knowledge;
- managing changing data and knowledge;
- nonstandard, multimedia and object-oriented data.

Databases with hundreds of fields and tables, millions of records, and multi-gigabyte size are quite commonplace, as well as terabyte (10^{12} bytes) databases. These datasets create combinatorially explosive search spaces for model induction and increase the chances that a data mining algorithm will find spurious patterns that are not generally valid. Methods for dealing with large data volumes include more efficient algorithms, sampling, approximation methods, massively parallel processing, dimensionality reduction techniques, and incorporation of prior knowledge.

Not only is there often a very large number of records in the database, but there can also be a very large number of fields (attributes, variables) so that the dimensionality of the problem is high. A high dimensional data set creates problems in terms of increasing the size of the search space for model induction in a combinatorially explosive manner. In addition, it increases the chances that a data mining algorithm will find spurious patterns that are not valid in general. Approaches to this problem include methods to reduce the effective dimensionality of the problem and the use of prior knowledge to identify irrelevant variables.

When the algorithm searches for the best parameters for one particular model using a limited set of data, it may *overfit* the data, that is it may model not only the general patterns in the data but also any noise specific to that data set, resulting in poor performance of the model on test data. Possible solutions include crossvalidation, regularization, and other sophisticated statistical strategies.

A problem related to overfitting occurs when the system is searching over many possible models. For example, if a system tests N models at the 0.001 significance level, then on average, with purely random data, $N = 1000$ of these models will be accepted as significant. This point is frequently missed by many initial attempts at KDD. One way to deal with this problem is to use methods which adjust the test statistic as a function of the search or randomization testing.

Missing and noisy data problem is especially acute in business databases. Important attributes may be missing if the database was not designed with discovery in mind. Missing data can result from operator error, actual system and measurement failures, or from a revision of the data collection process over time (e.g., new variables are measured, but they were considered unimportant a few months before). Possible solutions include more sophisticated statistical strategies to identify hidden variables and dependencies.

Hierarchically structured attributes or values, relations between attributes, and more sophisticated means for representing knowledge about the contents of a database will require algorithms that can effectively utilize such information. Historically, data mining algorithms have been developed for simple attributevalue records, although new techniques for deriving relations between variables are being developed.

In many applications it is important to make the discoveries more understandable by humans. Possible solutions include graphical representations, rule structuring, natural language generation, and techniques for visualization of data and knowledge. Rule refinement strategies also help address a related problem: the discovered knowledge may be implicitly or explicitly redundant.

A standalone discovery system may not be very useful. Typical in tegration issues include integration with a DBMS (e.g. via a query interface), integration with spreadsheets and visualization tools, and accommodating realtime sensor readings. Highly interactive human-computer environments as outlined by the KDD process permit both human-assisted computer discovery and computer-assisted human discovery. Development of tools for visualization, interpretation, and analysis of discovered patterns is of paramount importance. Such interactive environments can enable practical solutions to many real-world problems far more rapidly than humans or computers operating independently. There are a potential opportunity and a challenge to developing techniques to integrate the OLAP tools of the database community and the data mining tools of the machine learning and statistical communities.

Many current KDD methods and tools are not truly interactive and cannot easily incorporate prior knowledge about a problem except in simple ways. The use of domain knowledge is important in all of the steps of the KDD process. For example, Bayesian approaches use prior probabilities over data and distributions as one way of encoding prior knowledge. Others employ deductive database capabilities to discover knowledge that is then used to guide the data mining search.

Rapidly changing (nonstationary) data may make previously discovered patterns invalid. In addition, the variables measured in a given application database may be modified, deleted, or augmented with new measurements over time. Possible solutions include incremental methods for updating the patterns and treating change as an opportunity for discovery by using it to cue the search for patterns of change only.

A significant trend is that databases contain not just numeric data but large quantities of nonstandard and multimedia data. Nonstandard data types include non-numeric, non-textual, geometric, and graphical data, as well as non-stationary, temporal, spatial, and relational data, and a mixture of categorical and numeric fields in the data. Multimedia data include free-form multilingual text as well as digitized images, video, and speech and audio data. These data types are largely beyond the scope of current KDD technology.

3.3 Basic requirements

The design of a knowledge discovery framework aims at formally characterizing all aspects mentioned in section 3.2, and at providing both the theoretical and methodological grounds for the investigation of the upcoming issues.

A natural question (investigated in 3.1) is whether data mining can be put in the same methodological grounds as databases. Relational databases, in this respect, represent the paradigmatic example, where a simple formalism merges rich expressiveness and optimization opportunities. The set of mathematical primitives and their closure property allows to express a wide set of queries as composition of such primitives. The same formalism enables query execution optimization such as query decomposition, constraint pushing, advanced data structures, indexing methods. Thus, putting data mining in the same methodological grounds essentially means being capable of decoupling the specification from the execution of a data mining query. From an optimization perspective, the challenge is how to merge the efficiency of DBMS technologies with data mining algorithms, more specifically, how to integrate data mining more closely with traditional database systems, above all with respect to querying.

A further aspect to be investigated is the process-oriented nature of knowledge discovery. The overall KDD process is quite complex, and requires combining several forms of knowledge along with the cooperation among solvers of different nature. Analytical questions posed by the end user need to be translated into several tasks such as:

- choose analysis methods;
- prepare data in order to be processed;
- apply chosen methods to the data;
- interpret and evaluate obtained results.

Each of the above issues is concerned with a specific dimension and can, in principle, be described in a different language and accomplished in a different framework. However, using different languages/frameworks results in an *impedance mismatch* between dimensions, which can be a serious obstacle to efficiency and efficacy. A major challenge in building KDD frameworks concerns the smooth cooperation between different dimensions. Thus, a coherent formalism, capable of dealing uniformly with all dimensions, would represent a breakthrough in the design and development of decision support systems in diverse application domains. The advantages of such an integrated formalism include the ability to formalize the overall KDD process, and the possibility to tailor a methodology to a specific application domain.

Mining tasks and techniques are implemented by means of ad hoc algorithms, whose results in most cases are not directly useful for analysis purposes: they often need a tuning phase, in which they are interpreted and refined. Typically, when analyzing data one needs to derive good initial hy-

potheses to start the pattern search, to incrementally refine the search strategy and to interpret discovered knowledge.

In addition, most data mining tools and methods require deep technical and statistical knowledge by the data analyst, and a clear comprehension of the data. An analyst is usually not a KDD expert but a person responsible for making sense of the data using available KDD techniques. Since the KDD process is by definition interactive and iterative, it is a challenge to provide a high-performance, rapid-response environment that also assists users in the proper selection and matching of appropriate tools and techniques to achieve their goals. There needs to be more emphasis on human-computer interaction and less emphasis on total automationwith the aim of supporting both expert and novice users.

Even when results are clear and easy to understand, the interpretation and usefulness of such results may not be immediate. Thus, analyzing data requires a special framework to ease the burden of turning analytical questions into calls to specific algorithms and mining tasks, and to turn mining results into actionable knowledge.

From a methodological viewpoint, data mining can be seen as advanced querying:

> *given a set of objects of interest Σ and a set of properties $\mathcal{L}$, which are the properties within $\mathcal{L}$ of interest according to the search criteria q?*

An ideal data mining framework should respect the closure of a query language as a basic design paradigm. Furthermore, it should be capable of supporting the user in specifying and refining mining objectives, combining multiple strategies, and defining the quality of the extracted knowledge. Finally, it should act as an interface between analysts and underlying computational systems. Within a simple perspective, the framework should ease the process of describing the entities of interest and their properties, and the pattern language upon which data mining algorithms should rely on. However, there are several further problems which affect the knowledge discovery process. Such issues require a structured, formal approach, as opposed to the informal and ad-hoc approach which still nowadays describes the knowledge discovery discipline.

Fundamental aspects the advocated framework has to consider are:

- the various forms of knowledge to represent;
- the repertoire of analytical questions to support;
- the type of analytical users to address.

Several forms of knowledge may be represented. The source data or primary knowledge (i.e. the data to be mined), the background or domain knowledge (i.e. the rules which enrich data semantics in a specific context), the mined or extracted knowledge (i.e. the patterns or models mined from the source data).

Answering to specific analytical questions is closely related to the particular characteristics of data that have to be supported and the type of reasoning needed to express relations among such data, it also depends on which kind of patterns and models are expected.

As far as users are concerned, there are two distinct types of analytical profiles: the domain expert and the data mining expert. The domain expert has to be supported in specifying and refining the analysis goals by means of highly expressive declarative queries. On the contrary, the data mining expert masters the KDD process and aims at constructing complex vertical analytical solutions, so he/she has to be supported in specifying and refining the analysis goals by means of procedural abstractions to control the KDD process.

A combination of design choices, according to the options offered by the above three aspects, defines the requirements of a framework that has to support knowledge discovery. The multiplicity of such options highlights the complexity of the scenarios as well as explains the high number of existing proposals, both in research and in industry.

In defining a knowledge discovery framework, four categories of criteria have to be taken into account: functionality, usability, support of auxiliary activities and performance.

Functionality

Functionality is the inclusion of a variety of capabilities, techniques, and methodologies that enable the framework to cope with different data mining problems and domains. Such capabilities include

- algorithmic variety;
- algorithm modifiability;
- model validation;
- prescribed methodology;
- reporting;
- model exporting;
- result feedback.

First of all, the framework has to provide an adequate variety of mining techniques and algorithms, but the user must also have the ability to modify and fine-tune the modeling algorithms. In addition to model creation, model validation has to be supported: in order to avoid spurious results, the framework has to encourage validation as part of the overall step-by-step methodology. Mining analysis results have to be reported in a variety of ways, providing summary results as well as detailed results. Furthermore, it has to be possible to select actual data records that fit a target profile, and after a model is validated it has to be possible to export the model for external use.

Of primary importance, is the possibility to allow the results from a mining analysis to be fed back into another analysis for further model building,

in effect models extracted by data mining algorithms very often need to be further processed (i.e. combined with other models). In addition, extracted models can be applied on (new) data to predict features or to select data accordingly to the knowledge stored in the model. It has to be possible to query a model for predictions and, possibly, to test the answers against known values, in order to estimate model accuracy. Hence, a data mining API should allow for defining a new model, populating it by extracting knowledge from data, accessing the knowledge in the model and, for predictive models, predicting values exploiting model knowledge. Mining operations composability should also be pursued in the design of a middle-ware KDD language. Unfortunately, however, the highlighted limits of present standards for model representation and API do not allow for having composability with other KDD frameworks. This is somewhat different from what happens in the relational databases world, where transparent integration can be achieved in accessing/querying external data sources.

As far as data are concerned, an abstraction level concerning *logical data* is required, that is domains of data to be used as input to data mining operations in order to specify the type of usage of attributes in building and applying a mining model. For example, a classical distinction is made between discrete and continuous attributes: discrete ones include binary, nominal, categorical and ordinal values; continuous ones include interval-scaled and ratio-scaled values. Taxonomies (or hierarchies) are another logical data element. While from the physical point of view they are tables, from the logical point of view they model domain-knowledge by defining hierarchies exploited by data mining algorithms. Also, weights (or probabilities) are logical data elements that make sense in affecting the role of an attribute or of an input row in the construction of a model.

Initially, the logical level specification is completely induced by the physical representation of data, but typically logical meta-data is then refined by the user. Also, it is quite common that during the KDD process several different specifications are tried in order to test performance of extracted knowledge by varying weights, types, taxonomies. Hence, the system should allow for specifying logical meta-data (such as attribute type, taxonomies, weights) in addition to physical meta-data.

Anyway, data is not only an input to the KDD process, but also an intermediate output and a final output. As an example, a classiffication/regression model could be used to fill missing field values of corrupted buying transactions. An intermediate table is then necessary to store the recovered transactions, before using them further, e.g. before submitting them to a time series analysis algorithm for predicting future levels of purchases of a particular brand. A data repository should be available as a *data staging* area for storing input, output and intermediate data of the KDD process.

A data mining model represents knowledge extracted from data. As for data, extracted mining models (either the final or the intermediate ones of a complex KDD process) should be stored in an appropriate repository, for

subsequent analysis or application. For instance, a classification model may be visually presented to a domain expert, or it may be applied on unseen data to predict unknown classes, or it can be input to an incremental classification algorithm that revise it based on additional training data. A model repository should be available as a *model staging area* for storing input, output and intermediate models of the KDD process.

A proprietary binary representation for storing models can be adopted, providing API's for access from external programs, navigating models, issuing queries in proprietary query languages and for import/export in some interchange format. Alternatively, an industry standard for actual models representation as XML documents is the Predictive Model Markup Language (PMML [80]), currently being adopted as interchange format, even if it can't be considered the equivalent of data connectivity standards for for data mining. PMML consists of DTDs for a wide spectrum of models, including association rules, decision trees, clustering, naive-bayes, regression, neural networks. PMML only covers the core of data mining models, possibly missing some issues, offering an *extension* tag, allowing for including additional contents in the model representation. Anyway, in practice, two PMML-compliant systems exporting models in different formats may not yield interchangeable models. While PMML is becoming a primary standard, adopted by major commercial suites, it is worth noting that it does not cover the process of extracting models, but rather the exchange of the extracted knowledge.

Usability

Usability is accommodation of different levels and types of users without loss of functionality or usefulness. One problem with easy-to-use mining tools is their potential misuse. Since, KDD is a highly iterative process, practitioners typically adjust modeling variables to generate more valid models. As a consequence, not only should a tool be easily learned, it should also help guide the user toward proper data mining rather than *data dredging*. In addition, a good framework has to provide meaningful diagnostics to help debug problems and improve the output.

- user interface;
- learning curve;
- user types;
- data visualization;
- error reporting;
- action history;
- domain variety.

The user interface has to be easy to navigate and uncomplicated, and results have to be presented in a meaningful way. The tool needs to be easy to learn and easy to use correctly. Preferably, several perspectives for beginning, intermediate, advanced users or a combination of users, has to be available,

in order to well suit the tool for its target user type (analysts, business end users, etc...). Data and modeling results have to be presented in a suitable way, via a variety of graphical methods used to communicate information. Error reporting has to be meaningful: error messages has to help the user debugging problems. A history of actions taken in the mining process needs to be kept. The user has to be capable of modifying parts of this history and re-execute (part of) the script. It has to be possible to use the framework in several different industries to help solve a variety of different kinds of business problems.

Auxiliary tasks support

Auxiliary Tasks Support allows the user to perform the variety of data cleansing, manipulation, transformation, visualization and other tasks that support the KDD process. These tasks include data selection, cleansing, enrichment, value substitution, data filtering, binning of continuous data, generating derived variables, randomizing, deleting records, and so on. Since it is rare that a data set is truly clean and ready for mining, the practitioner must be able to easily fine-tune the data for the model building phase of the KDD process.

- data cleansing;
- data manipulation/filtering;
- data type flexibility;
- deriving attributes;
- data randomization;
- data sampling;
- record deletion;
- metadata manipulation.

Depending on the goals and requirements of the KDD process, analysts may select, filter, aggregate, sample, clean and/or transform data. Automating some of the most typical data processing tasks and integrating them seamlessly into the overall process may eliminate or at least greatly reduce the need for programming specialized routines and for data import/export, thus improving the analysts productivity.

As far as the acquisition phase is concerned (where access to physical data is performed), one can notice that very often data reside on databases. Hence, essential tools for gathering data from multiple sources are needed, such as database connectivity standards (e.g. ODBC or JDBC): the framework has to be capable of interfacing with a variety of data sources without any auxiliary tool. Data connectivity standards offer API's for connecting to a data source, issuing SQL queries, navigating returned record-sets, and accessing database meta-data. In many cases, the data to be analyzed is scattered throughout the corporation, it has to be gathered, checked, and integrated before a meaningful analysis can take place. The capability to directly access different data sources can thus greatly reduce the amount of data transforming.

The user has to be capable of modifying spurious values in the data set or perform other data cleansing operations, such as global substitution of one data value with another (e.g., replacing "M" or "F" with 1 or 0 for uniformity) or selection of subsets of the data based on user-defined selection criteria. Other filtering activities are for example concerned with handling blanks, which usually have to be replaced by a variety of derived values (e.g., mean, median, etc...) or user-defined values. A wide-variety of data types have to be supported, an important capability is binning of continuous data to improve modeling efficiency. This can be done either implicitly, or the decision can be left to user discretion. Some mining algorithms have restrictions upon the attribute types of the input data. For example, neural networks usually require all attributes to be of numeric type. Other approaches may not be able to handle continuous (real) data, etc.

The creation of derived attributes based on the inherent attributes has to be supported. A wide-variety of methods available for deriving attributes (e.g. statistical functions, mathematical functions, boolean functions, etc.) has to be provided. Randomization and sampling of data prior to model building are of great importance. Randomization and sampling have to be effective and efficient. It is also often required to delete entire records from entire segments of the population that may be incomplete or may bias the modeling results in some way, and sometimes these records have to be reintroduced later. Data descriptions, types, categorical codes and formulae for deriving attributes are also very useful, as well as the ability to manipulate tables' metadata.

Performance

Performance is the ability to handle a variety of data sources in an efficient manner. Hardware configuration has a major impact on tool performance from a computational perspective. Furthermore, some data mining algorithms are inherently more efficient than others. This requisite focuses on the qualitative aspects of a framework's ability to easily handle data under a variety of circumstances rather than on performance variables that are driven by hardware configurations and/or inherent algorithmic characteristics.

- data size;
- efficiency;
- robustness;
- software architecture.

Scaling linearly to large data sets is of paramount importance, results have to be produced in a reasonable amount of time relative to the data size and the limitations of the algorithm. Maximum number of tables/rows/attributes are are theoretical limitations on the processing capabilities of the discovery tool, as well as practical limitations that are posed by computing time, memory requirements, expressing and visualization capabilities and so on. A

tool that holds all data in main memory for example may be not appropriate for very large data sources, even if the theoretical maximum number of rows is unlimited. Once the analyses to be performed are defined, the tool has to be capable of running on its own, but being monitored and with the possibility of external intervention. If the tool cannot handle a data mining analysis, it is preferable to fail early and not when the analysis appears to be nearly complete. Once the analysis has started, the tool should run consistently and autonomously without crashing. Finally, it is desirable that the software run on a wide-variety of computer platforms, the possibility to choose among several software architectures (stand-alone, client-server, both) is also appreciated.

3.4 Current approaches

As discussed in the previous sections, nowadays most enterprises are actively collecting and storing large databases and many of them have recognized the potential value of these data as an information source for making business decisions. The aforementioned issues received a deep and recurring attention by data mining researchers, as well as data mining software producers. The dramatically increasing demand for better decision support is answered by an extending availability of knowledge discovery and data mining products, in the form of research prototypes developed at various universities as well as software products from commercial vendors.

The unifying goal is extracting highlevel knowledge from lowlevel data, finding understandable patterns that can be interpreted as useful or interesting knowledge. In order for a data mining tool to be able to cope with real world applications, of fundamental importance are also scaling properties of algorithms to large data sets. But the main observation is that in the current state of the art, the lack of support to knowledge discovery as an actual multi-step process makes impractical all those applications, that involve multiple stages of analysis and manipulation for both data and patterns, in which the results at each stage are required to become the input to the subsequent stages.

Until a few years ago, knowledge discovery tools were mainly used in research environments, but nowadays it is primarily the commercial developers who advance the application of these technologies to real business and scientific problems, this is witnessed by the large number of commercial tools and RDBMS offering KDD algorithms. Such sophisticated tools, which aim at the mainstream business user, are rapidly emerging. Although several data mining software tools currently share the market leadership, there is no single best tool for all data mining applications, and yet new tools keep entering the market.

Anyway, the business users demands on these tools continue to exceed the available technology. As the KDD field matures in response to these demands,

business users face the daunting task of deciding which tool best suits their needs and budgets. Currently the dollar cost of these tools is substantial, but the cost of selecting an improper data-mining tool for a particular application is even more costly in terms of personnel resources, wasted time, and the potential for acting on spurious results.

Currently available tools deploy either a single technique or a limited set of techniques to carry out data analysis, however, there is no best technique for data analysis. The issue is therefore not which technique is better than another, but rather which technique is suitable for the problem at hand. A truly useful tool has to provide a wide range of different techniques for the solution of different problems, since different techniques outperform each other for different problems. Some tools are based on the relational model and allow querying of the underlying database. Anyway, many tools just take their input in form of one table, where each sample case (record) has a fixed number of attributes, and operate separately from data sources, requiring a significant amount of time spent with data export, import, pre and post processing, transformation.

Currently available data analysis products generally fall into two categories.

- Drill-down analysis and reporting, provided by vendors of RDBMSs, sometimes in association with on-line analytical processing (OLAP) vendors. These systems provide a tight connection with the underlying database and usually deploy the processing power and scalability of the DBMS. They are also restricted to testing user provided hypotheses, rather than automatically extracting patterns and models.
- Stand-alone pattern discovery tools, which are able to autonomously detect patterns in the data. These tools tend to access the database off line; that is, data is extracted from the database and fed into the discovery engine. Many tools even rely on keeping all their data in main memory, thus lacking scalability and, therefore, the ability to handle real world problems. Additionally, these tools are often insufficiently equipped with data processing capabilities, leaving the data preprocessing solely to the user. This can result in repeated time-consuming import-export processes.

With the increasing number of proposed techniques as well as reported applications, it becomes clearer and clearer that any fixed arsenal of algorithms will never be able to cover all arising problems and tasks. It is therefore important to provide an architecture that allows for easy synthesis of new methods, and adaptation of existing methods with as little effort as possible. There are usually three stages at deploying KDD technology in an organization:

1. The potential of KDD is discovered. First naive studies are performed, often by external consultants (which are data mining specialists).
2. Once the profitability of KDD is proven, it is used on a regular basis to solve business problems. Users usually are teams of analysis experts

(with expertise in KDD technology) and domain experts (with extensive knowledge of the application domain).
3. Fully exploitation of KDD technology within the organization. End users are enabled to perform their own analysis according to their individual needs. Although widely still a vision, the necessity for this stage is clearly recognized.

Obviously the different users at these stages have different demands and also bring different prerequisites. Most of the available tools are aimed at analysis experts, requiring an unaffordable amount of training before being useful to novice end users. Typical end users are for example marketers, engineers or managers. These users are less skilled in complex data analysis and have less knowledge of the nature of the data available, but have a thorough understanding of their occupation domain. Furthermore, they are usually not interested in using advanced powerful technology, but only in getting clear, rapid answers to their everyday business questions.

End users need simple-to-use tools that efficiently solve their business problems. Existing software packages lack sufficient support for both directing the analysis process and presenting the analysis results in a user-understandable manner. If not, they are restricted to a very limited set of techniques and problems. Optimally, a better usability by novice users would have to be achieved without giving up other desirable features such as flexibility and/or analysis power.

In many applications, including the vast variety of nearly all business problems, the data is not stationary, but rather changing and evolving. This changing data may make previously discovered patterns invalid and hold new ones instead. Currently, the only solution to this problem is to repeat the same analysis process (which is also work-intensive) in periodic time intervals. There is clearly a need for incremental methods that are able to update changing models, and for strategies to identify and manage patterns of temporal change in knowledge bases.

Todays databases do not contain only standard data such as numbers and strings but also large amounts of nonstandard and multimedia data, such as free-form text, audio, image and video data, temporal, spatial and other data types. Those data types contain special patterns, which can not be handled well by the standard analysis methods. Therefore, these applications require special, often domain-specific, methods and algorithms. Object-oriented and nonstandard data models, such as multimedia, spatial or temporal, are largely beyond the scope of current KDD technology.

In conclusion, despite its rapid growth, KDD is still an emerging field, and the development of successful data mining applications still remains a tedious process. However, while there are fundamental problems that remain to be solved, there have also been numerous significant success stories reported, and the results and benefits are sometimes very impressive. Although the current methods still rely on fairly simple approaches with limited capabilities,

reassuring results have been achieved, and the benefits of KDD technology have been convincingly demonstrated in the broad range of application domains. The combination of urgent practical needs and the strong research interests lets us also expect a future healthy grow of the field, drawing KDD tools into the mainstream of business applications.

As far as open source and academic software are concerned, the most famous and distinguished data mining tool is presented below.

WEKA

Weka [94] is an open-source tool-bench for machine learning and data mining, implemented in Java. The algorithms provide a standard interface which makes them directly available within custom Java code. Weka main features include a comprehensive set of data pre-processing (filtering) tools, several learning algorithms for classification, regression, clustering, and association mining, together with model evaluation tools, and standard interfaces for filters, algorithms and evaluation methods, which can be hence customized to specific application needs.

The main strength of Weka lies in its flexibility: its specification as a library allows to model complex tasks within Java code, with the help of a clean, highly customizable, object-oriented class hierarchy for each element of interest in a knowledge discovery process. Clearly, the high flexibility in Weka also represents its weakness: there is no standard way of encoding and exploiting background knowledge, which is on the contrary demanded to the user/programmer. In particular, there are no standard mechanisms for reasoning on the extracted knowledge, which should be explicitly encoded and programmed.

Other leading vendors of data mining tools are cited in the following list.

Angoss Software

Angoss [86] is a data mining workbench provider and a traditional competitor to SPSS' Clementine or SAS' Enterprise Miner. Angoss has several customer sites, and many of its clients prefer specific elements of its solution (for example, the decision tree KnowledgeSeeker application). Angoss has a strong orientation and a considerable domain expertise toward the financial services (banking, insurance and mutual funds) and telecommunication industries. Support for implementation and ongoing use is also provided. Angoss offers the solution as either packaged software or an application service provider (ASP) solution.

Fair Isaac

Fair Isaac [57] is pursuing a broader strategy of enabling enterprise decision automation. The combination of its Blaze Advisor rule engine and analytical modeling tools can be used to create extremely sophisticated and complex automated decision processes for any business function. Fair Isaac can leverage a strong service business to complement its analytical offerings, spanning the range of services from hosting, to implementation, to a pure software sale. Analytic models developed with Fair Isaac's Model Builder software can be deployed directly into the production environments, where they'll be used via the company's Blaze Advisor rule engine.

IBM DB2 Intelligent Miner

Despite its name this product [70] is not dependent upon DB2 and may be used with other data sources. More recently, IBM introduced a set of three further DB2 Intelligent Miner products: Modeling, Visualizer and Scoring, all of which may be implemented separately. In particular, DB2 Intelligent Miner Scoring may be implemented not only in conjunction with Modeling (which requires DB2) and Visualizer, but also with DB2 Intelligent Miner for Data and with third party data mining products. Perhaps the most important difference between DB2 Intelligent Miner for Data and the other DB2 Intelligent Miner products is that the former was originally designed as a work bench style tool for environments where data mining was essentially regarded as a stand-alone exercise.

Infor

Infor [56] has a great deal of experience in CRM, with its Interaction Advisor product line. The acquisition of Infor CRM Epiphany by SSA Global, and then by Infor, provided more development resources and an extensive sales channel.

KXEN

KXEN [63] was founded with a vision of automating the rapid creation of large numbers of models, and building models in environments where there are thousands of potentially significant variables. As such, KXEN offers a significant amount of analytical differentiation from the established vendors as well as the other, smaller startups in the market. Although several vendors (such as Advizor Solutions, Alterian and smartFocus) embed KXEN into their applications, KXEN's focus has increasingly moved toward independent positioning as a best-of-breed data mining tool targeting the CRM space, rather than as a generic data mining toolset.

Oracle Data Mining

Oracle Data Mining [71] is an option to Oracle Database 11g Enterprise Edition. It enables the user to build and deploy applications that deliver predictive analytics and new insights. Application developers can build applications using ODM's SQL and Java APIs that automatically mine Oracle data. Data, models and results remain in the Oracle Database. Data analysts can quickly access their Oracle data using the optional Oracle Data Miner graphical user interface and explore their data to find patterns, relationships, and hidden insights. Oracle Data Mining provides a collection of in-database data mining algorithms. Anyone who can access data stored in an Oracle Database can access Oracle Data Mining results-predictions, recommendations, and discoveries using SQL-based query and reporting tools including Oracle Business Intelligence EE Plus.

Portrait Software

Portrait Software [87] has successfully capitalized on its acquisition of Quadstone from late 2005. Access to the Portrait sales channel and financial resources gives the Quadstone (now Interaction Optimizer) division greater viability and visibility. Portrait has retained the analytical and marketing orientation that Quadstone had prior to the acquisition. This should enable the ongoing development of Portrait's innovative applications, such as Uplift modeling (to distinguish offer-driven responders from those who would have responded anyway).

SAS

SAS [83] is the largest vendor in the overall data mining market. It has the most analysts, the most client experience and tends to be the standard tool with which data mining outsourcers and service providers must be familiar. As such, there's an unmatched "ecosystem" of talent and experience for SAS in the marketplace. With the most-complete set of data preparation and analytical tools in the market, there are few problems that SAS technologies can't solve. Enterprises seeking a "one-stop shop" platform for all analysis (particularly data mining) and supporting capabilities should consider SAS. Although much of SAS' recognition in customer data mining space is due to the popularity of its wide set of tool-based capabilities, the company is also delivering packaged applications to support customer data mining. These solutions have contributed to the breadth of experience and sophistication in SAS' customer base, providing plenty of best-practice examples of how to use SAS to support a customer analytics initiative.

SQL Server Data Mining and OLE DB

A data mining component [72] is included in Microsoft SQL Server 2000 and SQL Server 2005, one of the most popular DBMSs. Apart from a few algorithms, the main contribution of SQL Server Data Mining is the implementation of OLE DB for Data Mining. OLE DB for Data mining [88] is an industrial standard led by Microsoft and supported by a number of ISVs. It leverages two existing relational technologies: SQL and OLE DB. It defines a SQL language for data mining based on a relational concept. More recently, Microsoft, Hyperion, SAS and a few other BI vendors formed the XML for Analysis Council. XML for Analysis covers both OLAP and Data Mining. The goal is to allow consumer applications to query various BI packages from different platforms.

SPSS

SPSS [89, 25] has the broadest vision of the analysis of all types of data (behavioral, demographic, survey and unstructured). SPSS has one of the strongest visions for the emerging concept of the model management environment, which is a way of consolidating and managing the results of analyses from several data mining tools for subsequent deployment and evaluation. SPSS has spent the past several years acquiring analytical applications in the predictive (Clementine for data mining and DataDistilleries for real-time predictions) and nonpredictive (NetGenesis for Web site analytics, LexiQuest for text mining and Dimensions for customer surveys) markets. SPSS has combined these into a series of applications under the slogan, "Predictive Enterprise Services".

ThinkAnalytics

ThinkAnalytics' [91] products are based on an open platform with an open library of extensible components that can be combined to perform a variety of analyses. The models are deployed in ThinkAnalytics' Think Intelligent Enterprise Server, where they're available for any application (usually targeted at customer-facing applications, such as the call center or Web site) for real-time scoring. ThinkAnalytics focuses on operationalizing and embedding predictive analytics, and has also embedded a third-party rule engine to provide the capabilities of a real-time recommendation engine.

Unica

Unica's [92] origins in data mining left it with a tool that became a forerunner of today's trend toward the rapid building and testing of models. Although the market has significantly moved on, Unica's application is still a relatively straightforward way of achieving this basic analysis goal. Unica has moved its

development focus toward a series of markets adjacent to data mining. The result is that Unica now offers one of the broadest marketing suites in the market. Although the data mining module is a valuable component, beyond its mere existence, it doesn't represent a competitive advantage for Unica.

4

Defining a formal framework

The development of an effective data mining framework has been lately investigated from different perspectives, with two main objectives. On the one hand, focus is to provide an interface between data sources and data mining tasks: under this perspective, a data mining framework is seen as a standard mean for specifying data sources, patterns of interest and properties characterizing them. On the other hand, a data mining framework is meant to support the design of specific procedural workflows, which integrate reasoning on the mining results and possibly define ad-hoc evaluation strategies and activations of the data mining tasks. The underlying idea here is to define a general data mining framework (in which eventually a data mining query language could be embedded) in order to effective support the whole knowledge discovery process.

In this thesis, we follow this research developing a foundational model for the knowledge discovery process: the 2W Model, that enables progressive data-mining workflows and provide an underlying procedural semantic for the overall process. The 2W Model follows from an influential foundation for data mining, namely the 3W Model, originally introduced into [59] and subsequently refined by [18], where some theoretical issues arising from the idea of modeling the KDD process have been discussed.

In our vision, a KDD process can be described as a functional expression. Assume for example that a relational table $\mathcal{R}$ with schema $\mathcal{R} = \{A, B, C\}$ is available. Also assume that in addition to traditional relational-algebra operators for data preprocessing, two new operators are available, namely $\circledast_C(t)$, which extracts a decision tree from a table t, using C as class attribute, and $\bowtie_C (t, c)$, which given a table t and a classifier c (e.g. the aforementioned decision tree), creates a table t' which is identical to t but has an extra column C, where each tuple gets the value computed by the classifier. Then, the expression

$$t' = \bowtie_D (\circledast_C(\sigma_{B<3}(R)), \sigma_{B<3}(R))$$

represents a process where $\mathcal{R}$ is split in two partitions: the first partition contains all the tuples such that $B < 3$, the second partition contains all the tuples such that $B > 3$. Next, a decision tree is trained over the first partition, with C as the class target. Finally, a prediction is made over the second partition, by adding a new attribute D containing the result of the prediction.

4.1 The 3W Model framework

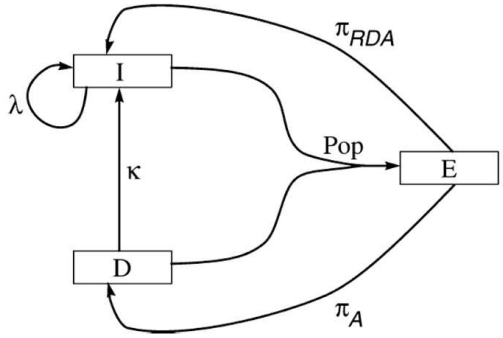


Fig. 4.1. The 3W Model

3W Model stands for *Three Worlds for data mining* [59, 18]:

- the D(ata)-world;
- the I(ntensional)-world;
- the E(xtensional)-world.

The D-World represents the raw data to be analyzed in terms of the basic entities of relational algebra, i.e. relational schemas and extensions. The attributes of such entities are associated with corresponding domains, that can be either categorical or numeric. Most activities, carried out in the preprocessing phase of a typical knowledge discovery application, can be modeled by means of specific operators of an extended relational algebra, that adds to the usual algebraic operators.

Objects in the I-World represent, instead, a particular class of data mining models, that is regions that can be defined as (sets of) conjunctions of linear inequality constraints on the attributes of the entities in the D-World. Starting from a set of basic regions, further regions can be formed via the definition of composition operators.

Since linear inequality constraints define regions in high dimensional spaces, such regions can be equivalently represented in an extensional way, as the set of all data points which satisfy those constraints.

In the E-World, a region is simply represented as an enumeration of all the tuples belonging to that region. Relations in this world are obtained by combining the relations of the two worlds previously defined, so that the schema of the resulting relation is the union of the schemas of some relation in the D-World and some other relation in the I-World. The algebra of the E-World is very limited and only involves those operators, that maintain the strong connection between regions and data tuples. Hence, aggregation, arithmetic and selection are the only world-specific operators available in the E-World.

Entities in the three aforesaid worlds can be related via suitable inter-worlds and intra-world operators:

- *Pop*: defines the extension of a model starting from its intentional representation and a relation;
- π: maps an extension to its counterpart in the corresponding worlds;
- κ: generates constraints from data constants;
- λ: generates new constraints by elaborating others.

Notice that, κ and λ are used to model a mining function. The generic mining operator κ extracts regions in the I-World from data in the D-World. These regions can be iteratively refined by means of the λ operator from the I-World to the I-World. The population operator *Pop* creates a relation in the E-World starting from some regions in the I-World and some other relations in the D-World. Finally, composite objects of the E-World can be projected to the other two worlds via the operators π_{RDA} and π_A, that allow to return in the I-World and D-World, respectively, via a simple selection of the proper attributes (data or constraints) within the E-World relation.

The 3W Model is mightily interesting for many reasons. Foremost, it provides a view of data mining in algebraic terms: a knowledge discovery process is the application of a sequence of operators in order to transform a set of tables. Furthermore, it is also fascinating from a methodological point of view: the object representation of 3W Model entities and the implementation of a suitable set of operators are key elements in the design of a powerful tool for knowledge discovery. However, some major limitations affect the 3W Model. In the D-World there is no possibility to express complex relations (i.e. cyclic relation), because the nesting of this data model has a fixed depth. Furthermore, a more serious limitation lies in the I-World, where regions are expressed by linear inequality sets. This means that fundamental mining models are not expressible, since their representations require more complex mathematic structures (i.e. SVM and clustering results, time point series, surrounding regions and so forth). The 2W Model in section 4.2 avoids both the foresaid limitations of the 3W Model. Indeed, it enables the description of complex objects and their properties and also supports the extraction all required patterns from raw data.

4.2 The 2W Model framework

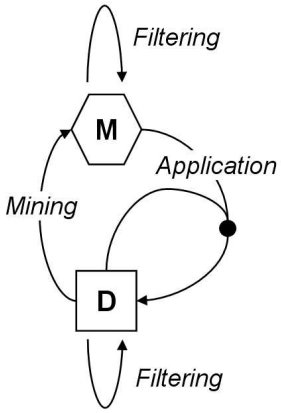


Fig. 4.2. The 2W Model

The 3W Model presented in section 4.1 has several strengths: the KD process is effectively modeled as a composition of operators, and the theoretic framework can be easily extended to the case of multi-relational data mining. However, the 3W Model also suffer from some main limitations:

- 3W Model is purely theoretical: no real-world data mining system could seriously rely on κ and λ operators, even if other embedded operators can also be imagined;
- 3W Model only supports basic data types;
- supported models have to be representable by means of sets of (linear inequality) constraints, that is only frequent patterns and decision rules are supported.

Amalgamating elements from different worlds causes an impedance mismatch, since the proposed formalization has to deal with different representations and different objectives. Anyway, separating the data and model worlds makes a lot of sense, and bridging such worlds through operators also introduces a nice interaction model. The basic idea beneath our vision is concerned with the extension of data and model worlds, through their equipment with some appropriate bridging operators.

In the 2W Model the essence of a knowledge discovery process is summarized as the interaction between two neatly divided worlds: the *data* world and the *model* world.

- *data* world represents the entities to be analyzed, with their properties and mutual relationship; raw data is organized in an object-relational format,

and attribute domains can be either primitive or complex object data types;
- *model* world offers an elegant and expressive framework for both exploratory analysis and reasoning; patterns concerning data entities, along with their properties and relationships are mapped to objects in the model world.

It is worth noticing that data objects are not necessarily tables composed by regions, instead it is possible to represent more complex structures. in addition, the κ operator is not predefined, rather it represents a template to extract a model from a table. Finally, models are generic objects, the *Pop* operator represents a template that can be instantiated in different ways depending on models classes (w.r.t. 3W Model, objects in E-World can be directly mapped to D-World, without an explicit representation of E-World).

As shown in fig. 4.2, data pre-processing and model post-processing are viewed as world-specific operations, whereas each intermediate pattern-mining step is considered as a suitable interaction relating entities in the two worlds.

This allows to formalize any knowledge discovery process as an algebraic expression, that is essentially a composition of operators representing (pseudo)elementary operations on the two worlds. There are three main kinds of operators for data and models:

- *Filtering* operators are self-injecting operations. They take a set of entities as input and produce a new set of entities, generating data from data (import and preprocessing purposes) and models from models (essentially post-processing). Within fig. 4.2, the data filtering and model filtering arrows denote such operations.
- *Mining* operators relate data entities to model entities, enable transitions from the D-World to the M-World. In practice, such operations correspond to the application of a data mining algorithm to a given data source. The result is a composite object, describing a pattern holding over such data sources.
- *Application* is a sort of opposite (join) operation w.r.t. mining function. In general, a model is a specification of a set of properties holding in the data. Applying a model to a data source essentially means making such properties explicit in extensional form: e.g., by associating each tuple in a table with the most likely target class according to the model, or by enumerating the frequent patterns appearing within the tuple. The only requirement is a decidable containment operator which each data mining model has to implement.

For the definition of the contours of the two worlds and their operators, one has to concentrate on which entities (i.e. which patterns) are supported in the model world, how data entities relate to model entities, and how constraint solving takes place. The formalization of such aspects strictly depends on the nature of the underlying applicative domain and pursued objectives. The

2W Model is a general model for the knowledge discovery process within any applicative setting.

Subsections 4.2.1 and 4.2.2 discuss, respectively, the D-World and M-World along with their specific operators, whereas section 4.2.3 describes mining and application operators, that allow worlds mutual interactions.

4.2.1 The D-World

The D-World represents the entities to be analyzed, as well as their properties and mutual relationships. Raw data is organized in an object-relational format. The D-World can be viewed as a database $\mathcal{D} = \{r_1(\mathcal{R}_1), \ldots, r_n(\mathcal{R}_n)\}$ of meaningful entities. The generic entity $r(\mathcal{R})$ is a relation with schema $\mathcal{R}$.

Formally,

$$\mathcal{R} = \{A_1 : Dom(A_1), \ldots, A_h : Dom(A_h)\}$$

where $A_1, \ldots, A_h$ correspond to descriptive attributes of the data within $r(\mathcal{R})$ and $Dom(A_1), \ldots, Dom(A_h)$ are their respective domains. Relation $r(\mathcal{R})$ is defined as $r(\mathcal{R}) \subseteq Dom(A_1) \times \ldots Dom(A_h)$. Attribute domains can be either primitive or object data types. Primitive types are assigned to simple features of the data and divide into categorical and numerical domains. Instead, object data types abstractly represent complex real-world entities as objects, equipped with application-dependant operations. Hereafter, we omit the specification of relation schema and use the resulting simplified notation to indicate an entity of $\mathcal{D}$. Furthermore, we denote by $t \in r$ a tuple of relation r and, also, exploit notation $t[A_i]$ to indicate the value of tuple t over a schema attribute A_i. The description of the D-World is general enough to be employed within any applicative setting.

Example 4.1 (`Paths` relation).
To elucidate, we introduce the reference relation `Paths`, that shall be used throughout this chapter to describe pedestrian and/or vehicle paths. Its schema attribute comprises id of type integer, type which takes on categorical values 'vehicle' and 'pedestrian', and path: an object data type, named *Event Collection*.

id	type	path
1	vehicle	$location_1^{t_1}, location_2^{t_2}, location_3^{t_3}$
2	vehicle	$location_2^{t_3}, location_1^{t_4}, location_3^{t_8}$
3	vehicle	$location_5^{t_1}, location_2^{t_2}, location_5^{t_5}, location_6^{t_7}$
4	pedestrian	$location_7^{t_1}, location_9^{t_5}$
5	pedestrian	$location_1^{t_1}, location_2^{t_6}, location_3^{t_9}$
6	pedestrian	$location_5^{t_5}, location_6^{t_9}$

The latter object type is used here to model the notion of a path, allowing to model paths evolving over time, and also providing a set of basic operations for manipulating paths data as well as answering topological and distance

queries. An in-depth coverage of the *Event Collection* data type will be given in section 5.2.1. In this example, given a tuple $t \in \texttt{Paths}$, $t[\mathsf{path}]$ is used to depict a collection of locations (represented by events) labeled with a time-stamp (a particular context of the corresponding event).
□

D-World operators

Data in the D-World are manipulated using *filtering* operators. For example, the usual (unary and binary) operators of traditional relational algebra such as $\rho, \sigma, \pi, \cup, \cap, \setminus$ and $\times$ are available, and also aggregation functions such as `SUM`, `COUNT`, `AVERAGE`, `MIN` and `MAX` are allowed to operate on collections of domain elements. Notably, aggregates are not relational algebra operators. In principle, they are used as parameters of some suitable aggregate formation operator. In our formalization, we express queries involving aggregates by means of suitably extended projection operators, in the spirit of the idea in [2], that allow the incorporation of aggregation functions into the algebra. Algebraic operators can be used to model suitable data preparation/manipulation tasks.

Example 4.2 (Data operators).
The composite operator $\pi_{\mathsf{path}}(\sigma_{\mathsf{type}\,=\text{'vehicle'}}(\texttt{Paths}))$ represents a trivial reduction of data size and dimensionality.
□

More complex preparation/manipulation tasks can be expressed by incorporating the basic operations of the (domain-specific) object-relational entities in the corresponding algebraic formulation, as shown in example 4.3.

Example 4.3 (Intersect operation).
A basic operation of the *event collection* data type is *intersect*, which queries whether two collections have an event in common, regardless its contexts values. Such a functionality can be exploited to filter from `Paths` and count all those routes that encounter, somewhere and at any given point in time, the route followed by a reference moving point having a specified identifier. To this purpose, by means of the expression $\texttt{T} = \rho_{\mathsf{route}\leftarrow\mathsf{path}}(\sigma_{\mathsf{id}\,=3}(\texttt{Paths}))$ one obtains a new answer relation `T` consisting of the route followed by the moving point with $\mathsf{id}{=}3$. Here, for convenience, the path attribute of `T` is renamed as route. `T` can now be joined with the whole content of `Paths` to find and count the desired routes, i.e. those paths that intersect the one in `T`. This can be expressed as $\pi_{\mathit{count}(\mathsf{path})}(\sigma_{\mathsf{path}.\mathit{intersects}(\mathsf{route})}(\texttt{Paths} \times \texttt{T}))$ where $\pi_{\mathit{count}(\mathsf{path})}(\cdot)$ is an extended projection operator that returns the size of the column path if it appears in the input relation, 0 otherwise. Notice that, the aforesaid extended projection operator can come in two flavors, depending on whether or not the path column is viewed as a bag.
□

Notably, queries over data is relevant because it permits to abstract away from the huge amount of punctual data, from this perspective, query formulation can be considered as closer to the way humans reason. Indeed, the D-World algebraic operators also play a fundamental role in post-processing data resulting from the application of a pattern in the M-World to raw route data.

4.2.2 The M-World

Patterns concerning data entities, along with their properties and relationships are mapped to objects in the model world, which provides an elegant and expressive framework for both exploratory analysis and reasoning. The M-World can be represented as a collection of objects $\mathcal{P}$, representing patterns unveiled at the different stages of the knowledge discovery process. Each pattern p is associated with an object-relational schema $\mathcal{R}$ and represents a (possibly infinite) relation r over $\mathcal{R}$. Intuitively, p represents a decision region over schema $\mathcal{R}$, so that a decidable operator $\vdash$ can be devised for bounding such a region.

Definition 4.4. *A pattern p is any (possibly infinite) set $\{t_1, \ldots, t_n, \ldots\}$ of tuples in $\mathcal{D}$ such that, for each $r \in \mathcal{D}$ and $t \in r$, the assertion $t \in p$ is decidable. We denote the property $t \in p$ as $p \vdash t$.*

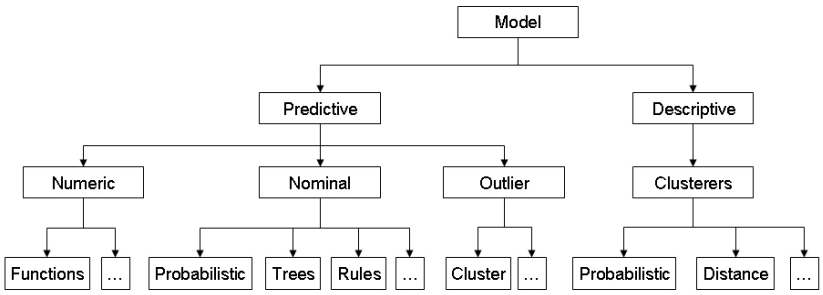


Fig. 4.3. A taxonomy of abstract model types in the M-World

As shown in fig. 4.3, the generic abstract data type **P** can be the prototype for any predictive or descriptive model. Hence, various instantiations are possible for the $\vdash$ operator. To elaborate on the latter aspect, we will exemplify next the semantics of the operation for some models of fig. 4.3.

Example 4.5 (Single-class Classifier).
Let informally define a pattern $p^{(pedestrian)}$ to represents all those tuples exhibiting instant speed less than $10Km/h$, which, according to the common sense, can be labeled as 'pedestrian'. Each moving object $t \in$ `Paths` can be

equivalently represented by its explicit route $(l_1, ts_1) \xrightarrow{speed_1} (l_2, ts_2) \cdots \xrightarrow{speed_{n-1}} (l_n, ts_n)$, i.e. a time-ordered sequence of locations l_i with respective time stamps ts_i, where $speed_i$ is the instant speed of the transition between locations l_{i-1} and l_i, which can be evaluated starting from locations' time stamps. Then, given a generic moving object t, $p^{(pedestrian)} \vdash t$, if $speed_i \leq 10Km/h$ for all $i \in [1, n-1]$.
□

Example 4.6 (T-pattern).
Let p be a pattern of the form $r_1 \xrightarrow{tc_1} r_2 \cdots \xrightarrow{tc_n} r_n$, where r_i is a spatial region and tc_i is a time constraint of the form $t_{min}^{(i)} \leq t \leq t_{max}^{(i)}$. A moving object $t \in \texttt{Paths}$ is recognized by p, i.e., $p \vdash t$, if t traverses all regions r_i in sequence, and the traversal time between r_i and r_{i+1} is within the time constraint tc_i.
□

Starting from the basic patterns in $\mathcal{P}$, we can define composite patterns as suitable combinations of patterns from possibly different prototypes. The individual instance of a prototype enumerates only data that results into a certain outcome when related to the pattern. However, in general, data can originate multiple outcomes, such as in the case of a pedestrian that traverses distinct T-patterns (see example 4.6). The notion of composite pattern is essentially an abstraction that allows to succinctly query multiple patterns of different prototypes for the recognized raw data.

Definition 4.7 (Composite pattern). *A composite pattern* $\mathbf{p}$ *is defined as follows:*

- $\mathbf{p} \equiv p$ *with* $p \in \mathcal{P}$ *is a composite pattern and the* $\vdash$ *operator is straightforwardly extended to* $\mathbf{p}$*, since* $\mathbf{p} \vdash t$ *coincides with* $p \vdash t$ *in this basic definition;*
- *the disjunction* $\mathbf{p}' \vee \mathbf{p}''$ *of two composite patterns* $\mathbf{p}'$ *and* $\mathbf{p}''$ *is still a composite pattern and the* $\vdash$ *operator is defined as* $\mathbf{p}' \vee \mathbf{p}'' \vdash t$ *if and only if either* $\mathbf{p}' \vdash t$ *or* $\mathbf{p}'' \vdash t$*;*
- *the conjunction* $\mathbf{p}' \wedge \mathbf{p}''$ *of two composite patterns* $\mathbf{p}'$ *and* $\mathbf{p}''$ *is still a composite pattern and the* $\vdash$ *operator is defined as* $\mathbf{p}' \wedge \mathbf{p}'' \vdash t$ *if and only if both* $\mathbf{p}' \vdash t$ *and*$\mathbf{p}'' \vdash t$*;*
- *the negation* $\neg\mathbf{p}$ *of a composite pattern is still a composite pattern, where* $\neg\mathbf{p} \vdash t$ *is it is not the case that* $\mathbf{p} \vdash t$*.*

In practice, a composite pattern is an abstraction for representing a new decision region that follows from the ones associated to individual patterns.

Example 4.8 (Composite pattern).
Let $p^{(pedestrian)}$ be the predictor defined in example 4.5, and p', p'' two further temporal annotated patterns introduced in example 4.6. The composite pattern $\mathbf{p} = p^{(pedestrian)} \wedge (p' \vee p'')$ is a pattern characterizing all pedestrians

traveling with an instant speed less than $10Km/h$ and traversing either p' or p''.
□

With an abuse of notation, in the following we shall assume that $\mathcal{P}$ is extended to contain both singleton and composite patterns. The latter can be further classified into local or global patterns, depending on whether such patterns are capable of recognizing each tuple in the associated domain.

Definition 4.9 (Global/local pattern). *Let* $\mathbf{p}$ *be a (composite) pattern and* $\mathcal{R}$ *an object-relational schema.* $\mathbf{p}$ *is said a global pattern w.r.t.* $\mathcal{R}$ *if and only if, for each relation* r *over* R *and each* $t \in r$*, it holds that* $\mathbf{p} \vdash t$*. Otherwise,* $\mathbf{p}$ *is said a local pattern.*

Example 4.10. Let us consider the pattern $p^{(vehicle)}$, which recognizes tuples representing routes exhibiting an instant speed greater than $10Km/h$. The pattern $p^{(vehicle)} \vee p^{(pedestrian)}$ is a global pattern for `Paths`, as it recognizes each tuple in the relation.
□

As stated before, a generic abstract model type $\mathbf{P}$ (see fig 4.3) can be associated to a given pattern p, depending on whether the decision region characterizes the pattern as a predictive or descriptive. In practice, a prototype $\mathbf{P}$ enumerates a subset of $\mathcal{P}$, such that some attributes $\varphi_1^{(\mathbf{P})}, \ldots, \varphi_n^{(\mathbf{P})}$ can be associated to each $p \in \mathbf{P}$. Formally, a prototype associates each pattern with a relational schema, where some specific pattern properties can be specified.

Definition 4.11. *Let* $\mathcal{R}$ *a relational schema. A* pattern schema $\boldsymbol{P}$ *on* $\mathcal{R}$ *is defined as* $\mathbf{P} = \{\mathcal{S} \times r | \mathcal{S} \subseteq \mathcal{P}, r \in \mathcal{R}\}$*.* $P \in \mathbf{P}$ *is defined a* pattern instance*.*

Example 4.12 (Classifier).
Assume that $\mathcal{O} = \{pedestrian, vehicle\}$ is a set of class labels denoting two alternative types of routes. Then, $\mathbf{Classifier}^{\mathcal{O}}$ is any subset of $\mathcal{P} \times \mathcal{O}$. In practice, each pair $\langle p, c \rangle \in \mathbf{Classifier}^{\mathcal{O}}$ represents a decision region denoted by p, whose tuples are associated to class c. For example, the tuple $\langle p^{(pedestrian)}, pedestrian \rangle$, where $p^{(pedestrian)}$ is defined in example 4.5) represents the pattern which classifies a path as pedestrian if its ground instant speed is below $10Km/h$.
□

M-World operators

We can extend the usual definitions of the σ and π (relational) operators, as follows. Let P be a pattern instance of $\mathbf{P}$, then:

- $\sigma_E(P) = \{\langle p, t \rangle \in P | E(t)\}$ where E is any boolean expression over the schema $\mathcal{R}$;

- $\pi_X(P) = \{\langle p, \pi_X(t)\rangle | \langle p, t\rangle \in P\}$ where $X \subseteq \mathcal{R}$.

Example 4.13 (Pattern schema).
Assume that prototype $\mathbf{T-pattern}$ is a pattern schema consisting of three attributes, namely *Pattern*, *Length* and *Type*, that associates each temporal annotated pattern in $\mathcal{P}$ with two characteristic features, respectively the length of the pattern in terms of consecutive spatial regions and the kind of routes that traverse it. Given the below instance P of $\mathbf{P}$

Pattern	Length	Type
p_1	5	vehicle
p_2	2	pedestrian
p_3	3	mixed
p_4	4	vehicle
p_5	2	pedestrian
p_6	3	mixed
$p_2 \vee p_5$	2	pedestrian
$p_3 \wedge p_6$	3	mixed

the intra-world operators $\sigma_E(P)$ and $\pi_X(P)$ can be used to suitably manipulate P. For instance, $\sigma_{Length \geq 4 \wedge Type='vehicle'}(P)$ selects those patterns in P (i.e. p_1 and p_4) that consist of at least 4 spatial regions and are traversed only by vehicles. In the selected fragment of P, the *Type* feature assumes a uniform value, thus becoming uninteresting. Thus, it can be filtered by means of the projection operator $\pi_{Pattern,Length}(\sigma_{Length \geq 4 \wedge Type='vehicle'}(P))$
□

4.2.3 Inter-worlds operators

Patterns are originated into the M-World from the raw data in the D-World via a *mining* operator. Such patterns are in turn used to inject new data into the D-World via the *application* operator. Mining and application operators formalize suitable interactions between data within the D-World and patterns in the M-World. Such interactions are the basic building-blocks in the definition of a knowledge discovery workflow.

The population of the M-World starting from the raw data in the D-World is performed through the *mining* operator.

Definition 4.14 (Mining operator). *Let $r \in \mathcal{D}$ be a relation from which to extract suitable patterns. The inter-world mining operator is defined as $\kappa : \mathcal{D}^k \rightarrow 2^{\mathcal{P}}$. κ represents a generic mining scheme, that receives a certain number of input relations and instantiates a mining model (i.e., multiple patterns) of a particular prototype (that is a pattern schema).*

Example 4.15 (Mining operator).

Assume that $\mathcal{D} = \{\texttt{Paths}\}$, where `Paths` is the reference relation of example 4.1. If κ represents the $T - pattern$ mining scheme defined in example 4.6, $\kappa(\texttt{Paths})$ results into an instance $T - pattern$ of a prototype $\mathbf{T - pattern}$.
□

Once accomplished the forward population of the M-World with the required patterns, these can be employed in the opposite direction, i.e. to backwardly populate the D-World with further data. Interestingly, this does not involve the explicit representation of further (composite) objects as in the E-World of the 2W Model. More simply, the raw data that falls within the decision region of a certain pattern is accumulated in the D-World as new data. The inter-world *application* operator $\ltimes\colon \mathcal{P} \times \mathcal{D} \rightarrow \mathcal{D}$, is the basic step of the application process.

Definition 4.16 (Basic application operator). *Let p be a pattern in $\mathcal{P}$ and r a relation in $\mathcal{D}$ over an object-relational schema $\mathcal{R}$. The basic application operator $p \ltimes r$ yields a new relation including each tuple $t \in r$ within the decision region of p. Formally,*

$$p \ltimes r \triangleq \{t \in r | p \vdash t\}$$

Clearly, the resulting relation $p \ltimes r$ is still an instance of $\mathcal{R}$.

Example 4.17 (Basic application operator).
Let $\mathcal{D}$ and $\mathcal{P}$ be respectively the D-World and M-World of Example 4.15. If p is a pattern over the $\mathbf{T - pattern}$ prototype, the expression $\texttt{Paths}' \triangleq p \ltimes \texttt{Paths}$ results into $\mathcal{D}' = \{\texttt{Paths}, \texttt{Paths}'\}$, where `Paths'` is a new relation including those moving entities from `Paths`, whose routes traverse the temporal annotated pattern p. In practice, $\mathcal{D}'$ represents the population of the original D-World with the new data `Paths'`, whose schema is identical to the one of the `Paths` relation.
□

The application operator can be straightforwardly generalized to deal with a composite pattern.

Definition 4.18 (Extended population operator). *Let $\mathbf{p}$ be a composite pattern and r a relation in $\mathcal{D}$ over an object-relational schema $\mathcal{R}$. By abuse of notation, we define the extended population operator $\ltimes\colon 2^{\mathcal{P}} \rightarrow \mathcal{D}$ as follows*

$$\mathbf{p} \ltimes \boldsymbol{r} \triangleq \{t \in r | \mathbf{p} \vdash r\}$$

Relation $\mathbf{p} \ltimes \boldsymbol{r}$ is again an instance of $\mathcal{R}$.

The above expression yields an enumeration of all the tuples in r that fall within the composite decision region $\mathbf{p}$.

Example 4.19 (Extended population operator).
Again, let $\mathcal{D}$ and $\mathcal{P}$ be respectively the D-World and M-World of Example 4.15. Moreover, assume that $\mathbf{p} = p_i \vee p_j$ is a composite pattern such that both p_i and p_j are two patterns from the `Paths` relation. The below expression $\texttt{Paths}'' = \mathbf{p} \bowtie \texttt{Paths}$ populates the original D-World with a further $\texttt{Paths}''$ relation, that enumerates all moving objects of `Paths` that traverse either p_i or p_j
□

Besides the application operator, a pattern identification operator can also be defined, that is dual with respect to the former.

Definition 4.20 (Pattern identification operator). *Assume that* $\mathbf{P}$ *is a prototype and* P *an instance of* $\mathbf{P}$*. Let* S *be any subset of* P*. The pattern identification operator* $\diamond : 2^P \times \mathcal{D} \rightarrow 2^P$ *is defined as follows:*

$$S \diamond r = \{p \in S | \exists t \in r : t \in p \bowtie r\}$$

In practice, the pattern identification operator queries a homogeneous pattern collection S for those models that recognize certain raw data.

Example 4.21 (Pattern identification operator).
One may ask which are the temporal annotated patterns from pedestrian routes, that are also traversed by vehicles. To this purpose, let $\texttt{Paths}' = \pi_{\text{type='pedestrian'}}\texttt{Paths}$ and $\texttt{Paths}'' = \pi_{\text{type='vehicle'}}\texttt{Paths}$ be the two required partitions of the `Paths` relation. If `T-pattern` is an instance of the $\mathbf{T-pattern}$ prototype, that consists of the patterns in $\texttt{Paths}'$, the foregoing query can be expressed via the following expression $\texttt{T-pattern} \diamond \texttt{Paths}''$
□

As a final remark, we emphasize that pattern-identification and application operators can be suitably combined to express complex analytical queries.

Example 4.22 (A complex analytical query).
With respect to the setting of Example 4.21, one may further ask which are the individual vehicle paths that traverse the pedestrian T-patterns. This query can be expressed as $(\texttt{T-pattern} \diamond \texttt{Paths}'') \bowtie \texttt{Paths}''$
□

4.2.4 Discussion

In [18], the authors study the expressiveness of the underlying model subject to different choices for the operators. In particular, there are some specific operators which make the resulting algebra computationally complete. A major limitation in the proposed approach is in the I-World, which is populated, as mentioned, by linear inequality constraints. This limitation means that the results of some data mining operations might not be expressible, as they require

more complex mathematical objects. Each world has its own entities, together with properties and relations: e.g., relational tables and operators in the data world, and conjunctions of linear inequality constraints in the model world. Now, since linear inequality constraints specify regions in high-dimensional spaces, such regions can be equivalently represented in an equivalent extensional way, as the set of all data points which satisfy those constraints.

The 2W Model introduces several meaningful differences w.r.t. the 3W Model:

- Entities in the M-World can represent any required patterns, even if with a mathematically complex structure, whereas I-World models correspond to simple regions, expressible via linear inequalities on the data attributes.
- In the 2W Model *mining* (κ) is not predefined and acts as a template to extract a model from a table. Currently, within the proposed formalization, the *mining* operator is instantiated by several different mining operators, according to the underlying applicative requirements. In this respect, a further improvement would be using a limited set of basic operators and derive all others from these via composition.
- In the 2W Model objects in the E-World are directly mapped to counterparts in the D-World, without an explicit representation in the E-World. By definition of population operator, the application of any model to the data in a relation of the D-World always produces a further relation within the D-World. This ensures that mining results can be progressively analyzed on a par with raw data via further manipulations, as exemplified next.

Example 4.23 (Modeling KD workflows).
Consider the case where one wishes to uncover the groups objects that move close to each other within a certain pattern. In such a case, patterns are first extracted into the M-World via a specific mining operator κ from the `Paths` relation. This results into an instance κ(`Paths`), that groups all the unveiled patterns. The latter are then treated on a par with data, to the purpose of identifying the paths inside the required pattern, which is accomplished by means of the inter-world application operator. Clusters can then be discovered in the required pattern, by applying a second mining operator κ' to the newly obtained data. In the 2W Model, the algebraic formulation of the aforesaid knowledge discovery workflow is $\kappa'(p \bowtie \texttt{Paths})$ where $p \in \kappa(\texttt{Paths})$ is the pattern to investigate for moving clusters, that can be chosen by means of specific intra-world operators. The above expression reveals the fundamental role of the application operator in the definition of a knowledge discovery workflow. Indeed, the operator enables the progressive and seamless discovery of further patterns in the data resulting at the end of a previous analytical process.
□

Finally, we recall that the D-World operators contribute to the expressiveness of the 2W Model framework, by playing a twofold role. On the one hand,

such operators can be used to represent preprocessing tasks, e.g. the reduction in size and/or dimensionality of the available data. On the other hand, they are useful for postprocessing purposes, such as in the act of filtering interesting patterns.

5

Realizing the formal framework

The aim of this chapter is to provide a concrete solution to the definition of a knowledge discovery framework, where the main purpose is to drive the user in defining and executing a *workflow* which represents a data mining process.

5.1 A 3-perspectives viewpoint

The framework we propose is a graphical environment for designing and managing KD processes. KD processes are defined by means of workflows, whose elements are node of a graph connected by directed edges. Graph nodes can be either tables, models or tasks. To put it simply, tasks are operators that allow transitions between the other two entities (tables/models), the generation of a table (or model) from another table is achieved by means of a task node. Workflow execution consists on sequential or parallel execution of tasks nodes, which are the connections between tables and models.

The defined framework includes the following functionalities:

- intelligent support to design workflows representing KD processes;
- execution and storing of generated workflows or subsets of components;
- visual environment to perform data analysis;
- visual environment to perform model analysis.

The general architecture of the system is shown in fig 5.1. We can see three basic kinds of modules, namely *CORE*, *GUI* (*Graphical User Interface*) and several other components that can be plugged into the system: *tasks*, *tables*, *models*, *table visualizers*, *models visualizers* and *bridges*.

The *CORE* module represents system kernel, *tasks*, *tables* and *models*, are realized as plugins, and represents workflow nodes (i.e. steps of the KD process). The *CORE* module exposes a rich set of API by which those plugins can be connected to each other, so KD processes are defined by a directed acyclic graph (DAG), whose components are plugins. *Table visualizers* and *models*

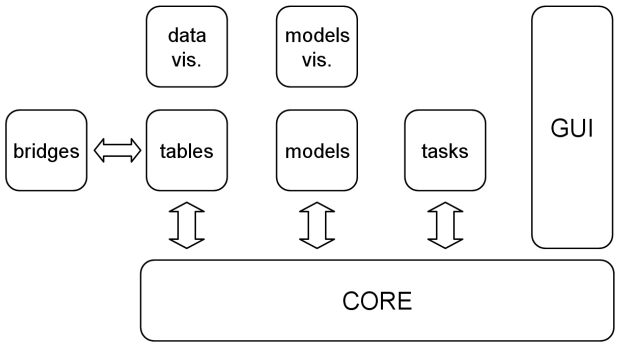


Fig. 5.1. System architecture

visualizers are also plugins, but they are not present in workflow composition, instead they are strictly connected to *tables* and *models* respectively, and are exploited to perform data and model explorations, as discussed in section 5.4. The *bridges* components are strictly related to tables, and represents a place where to store tables' data (a detailed discussion is made in section 5.2). Finally, the graphical user interface (*GUI*), which is developed upon a series of API exposed by *CORE*.

According to the above definitions, it comes out that a main feature of the proposed framework is its *extensibility*, since the system can be enriched by user-defined plugins, which can be implemented to accomplish ad-hoc needs and easily plugged into the system.

Another important feature of the proposed framework is its *easy of use*. The workflow is designed exploiting a graphical editor, which makes it simple to create, position, move and connect the aforementioned elements. User interaction takes place according to three different perspectives:

- **workflow perspective**: used to create, configure and connect nodes;
- **data perspective**: allows to perform data exploration;
- **model perspective**: allows a graphical representation of models.

Picture on figure 5.2 shows the workflow perspective, data and model perspectives are shown in figure 5.3 and better examined in section 5.4. The workflow represented in fig. 5.2 is composed by an acquisition task, a table, a mining task and the corresponding generated model. As can be seen, different elements are typified with different shapes.

By means of workflow view, user can design a KD process selecting and connecting nodes. Placing a node into the workflow is carried out by means of drag and drop operations from a node repository which contains all available nodes (i.e. a table node and all available models and tasks) to the work area (i.e. the place where the workflow is composed). When a node is placed on

the work area, a new *instance* of the node will appear. Instantiated nodes can be moved, resized, renamed or deleted, those operations are accomplished by drag and drop or by selecting corresponding commands on some appropriate menu that will come out right-clicking on the node. Nodes can be connected joining source and destination ports with a drag and drop operation.

Fig. 5.2. Workflow view

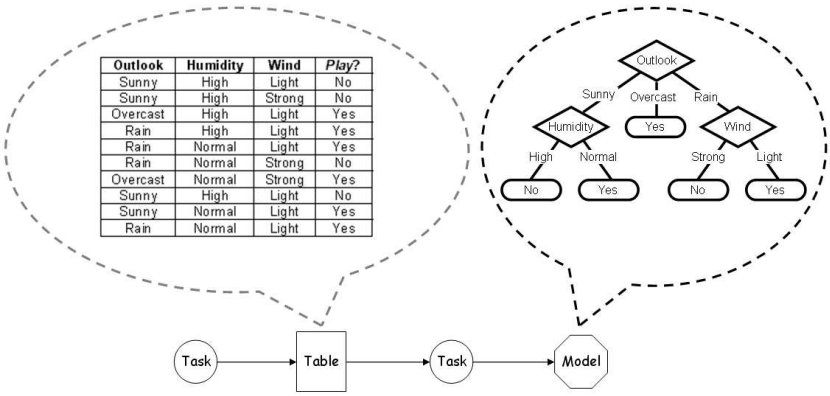


Fig. 5.3. Data and model views

In the node repository, nodes are organized into several categories, as specified in the following schema.

- Models: contains mining algorithms divided in several subcategories (see fig. 4.3):
  - *predictive*: contains all predictive models, grouped by their characteristic;
  - *descriptive*: contains all descriptive models grouped by their characteristic;
  - *dependency*: contains all dependency models grouped by their characteristic;

Instantiated nodes represent models generated with the specific corresponding algorithm. Specific features of each mining models group was previously presented in section 2.2.

- Tables: contains a sole table object, responsible for data storage during preprocessing and mining operations. A table can be inserted in the work area in two ways: dragging the aforementioned node (like other nodes) or directly drawing a connection from a task output port. The second option works for all tasks that can *guess* their output tables structure before their execution. A discussion aimed at explaining the way tables are capable of backing up data using either main memory or external sources, is done in section 5.2.
- Tasks: contains several subcategories:
  - *acquisition*: contains tasks that allow to import data from external sources, as well as models saved in previous sessions;
  - *mining*: contains tasks to extract a model from a table, eventually testing it on the fly;
  - *application*: contains tasks used to apply previously generated models to tables;
  - *filters*: contains tasks to handle and transform tables;
  - *export*: contains tasks that allow to export data and models in various formats. A detailed presentation of task is done in section 5.3.

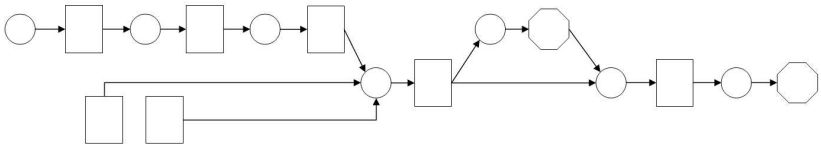


Fig. 5.4. A complex workflow

A typical utilization of the framework by means of the graphical user interface takes place in four phases, which are typically iterated many times:

- knowledge discovery process composition;
- partial or complete execution of generated workflow;
- exploration and visualization of generated tables and models;
- generation of new data by means of application of generated models.

5.2 The data retention dilemma

Bridges and tables are used to store and manage data. Bridges are not part of a workflow, then cannot be inserted in the workflow-perspective work area,

they only represent the physical device used to store data accessed via tables nodes. In a sense, a bridge represents the connection between logical and physical representation of data, and can be seen as a tables' container. More specifically, a table is always connected to its own bridge, and a bridge can contain many tables.

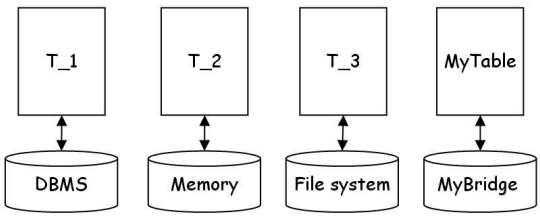


Fig. 5.5. Tables and bridges

Two straightforward examples of bridges are *MemoryBridge* and *DBMSBridge*. A *MemoryBridge* is responsible for providing support to *MemoryTables*, backing up tables' data into system main memory. A *DBMSBridge* bridge is responsible for providing support to *DBMSTables*, backing up table's data in an external DBMS. Let's clarify this concept saying that a *DBMSBridge* contains required logic for DBMS connection, whereas a *DBMSTable* is capable of accessing its own data (which physically reside on a DBMS) via the underlying *DBMSBridge*.

As can be seen from figure 5.6, once a table has been created the whole table-bridge mechanism remains behind the scenes, and from a user's perspective, data can be written and read row by row using the same conventional methods for all kinds of tables, regardless of the particular associated bridge.

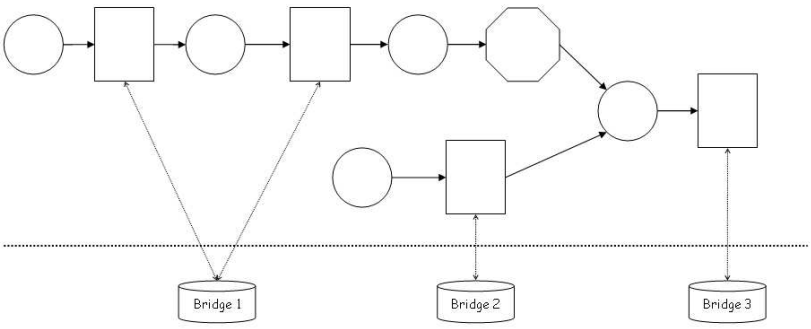


Fig. 5.6. Abstract and physical representation of data

Table structure is defined by means of metadata, used to specify table's attributes which define table's columns types. A table metadata object contains a collection of attributes, whereas table content consists of a collection of rows, whose values are interpreted according with the corresponding attribute type specified in table metadata.

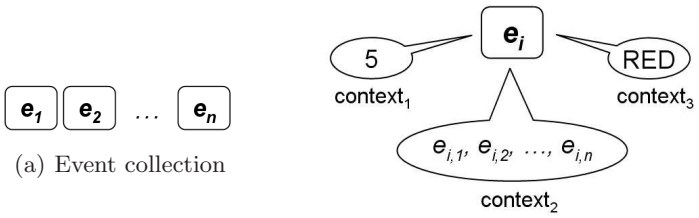
The following attribute's types are supported:

- *integer*: to represents integer values;
- *double*: to represents numeric values which can vary in a continuum;
- *nominal*: to represents a set of possible values;
- *string*: to represents free text;
- *event collection*: to represents collection of *events* where an *event* is a complex object.

The definition of integer, double and string attributes is straightforward. Nominal attributes are used when values can be chosen among a set. When defining a nominal attribute, a list of all possible values *can* be specified, in effect it is not always possible to know which values an attribute will take when specifying table metadata (for example, in acquisition tasks), so when writing on a table, if a new entry is seen for a given nominal attribute value, the entry is added to the set. Event collection attributes are described in details in section 5.2.1.

5.2.1 Dealing with complex data: `events`

Working with real-world applications, ordinary attributes types are not always suitable to model all kind of data. In order to overcome this issue, our implementation of the KD framework is equipped with a complex data type: event collection.



(a) Event collection

(b) Event

Fig. 5.7. Model visualization

As can be seen in figure 5.7, an event collection is a collection of complex objects named events (see 5.7(a)). In turn, an event is an object that has

a name (the context name), and a collection of properties named contexts (see 5.7(b)). Each context holds a value, which can be either a basic data type or an event collection.

Event collection attributes can be used, as an example, to model textual data. A textual document can be easily mapped to an event collection of terms. Each term (i.e. a string from the document mapped to an event) will have a set of properties (i.e. event's contexts) such as: string length, string category, frequency within the document, and so on. Modeling textual data this way, one can easily perform data exploration via statistics on the aforementioned properties, as well as performing data filtering operations such as stop-words removal, non frequent terms identification, etc.

5.3 Implementing the 2W algebra: tasks

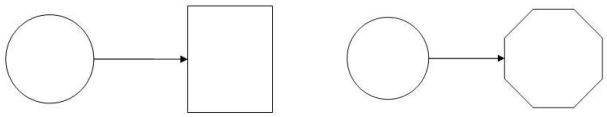
Data pre-processing, models post-processing, mining and application operators can be devised as functions having a domain in a given world and a codomain into the same or another world. In section 4 a set of basic operators have been proposed, discussing the expressiveness of the resulting model.

In our proposed implementation, those basic operators are modeled by means of tasks objects. Tasks are operators capable of manipulating and transforming tables and models. In practice, a task can produce/populate one or more output tables or/and models, on the basis of one or more input tables or/and models. Two other kinds of task which have no input/output component respectively are introduced for data/model import/export, namely data acquisition, model acquisition, data export and model export. Tasks can thus be divided in:

- acquisition: import tables/models from external sources;
- mining (i.e. mining algorithms): allow transitions from tables (data-world) to models (model-world);
- application: join operators that allow transitions from model-world to data-world;
- data-filters: data pre-processing;
- model-filters: model post-processing;
- export: export tables/models to external destinations.

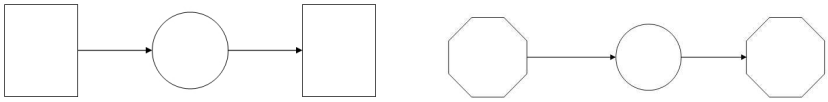
Tasks can be configured using specific parameters, for example, an acquisition task in charge of importing a text file, needs to know the path where to find the particular text file, or still, a task capable of transforming somehow a table attribute, needs to know which particular attribute of the input table needs to be transformed.

Figure 5.8 shows a table acquisition task (see 5.8(a)) and a model acquisition task (see 5.8(b)). Those kinds of tasks have no input and exactly one output, of type table and model respectively.



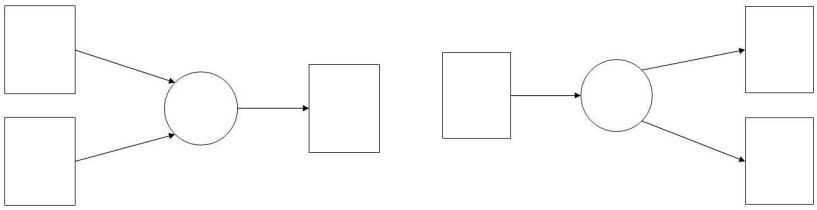
(a) Table acquisition (b) Model acquisition

Fig. 5.8. Acquisition tasks



(a) Table filtering with 1 input and 1 output

(b) Model filtering with 1 input and 1 output



(c) Table filtering with 2 inputs and 1 output

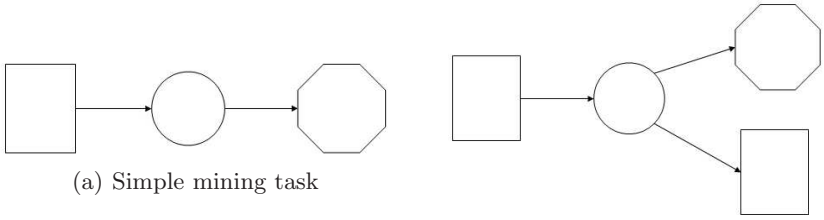
(d) Table filtering with 1 input and 2 outputs

Fig. 5.9. Filtering tasks

Figure 5.9 shows several filter tasks. Table filtering tasks (see 5.9(a) 5.9(c) 5.9(d)) can have an arbitrary number of tables/models in input or output. As an example, data-filtering tasks are used for performing SQL-like operations on tables, sampling tables, transforming attributes types (e.g. from numeric to nominal), discretizing attributes, and so on. Model-filtering tasks (see 5.9(b)) are used for example to prune a tree using some given criteria, to reduce a set of rule basing on a minimum support/confidence, and so on.

Figure 5.10 shows three kinds of mining tasks, namely simple mining (see 5.10(a)), hold out and cross validation (see 5.10(b)). The simple mining task has one input table and one output model, and its only responsibility is to build the output model using the whole input table.

The hold out mining task has an input table and two outputs: a model and a table. This task is capable of building the output model upon a fraction only of the input table (the so-called training set), the remaining data (the so-called test set) are used to test model accuracy. In effect, the output table



(a) Simple mining task

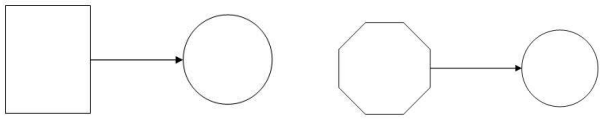
(b) Hold out and cross-validation mining tasks

Fig. 5.10. Mining tasks

is filled with tuples used to test the model, having the same structure of the input table, plus one extra column which represents the outcome of the inferred model (e.g. the class value or the cluster label). The fraction of data to be used as training/test set is fixed by a task's parameter.

The cross validation mining task is similar to the hold out mining task, since it is used for building a model and at the same time evaluating it. Anyway, this task does not perform a simple split between training and test set, instead an n-fold cross-validation technique is used (see [74]). The number of folds is fixed using a specific task's parameter.

The application task has two inputs: a model and a table, and one table as output. This task is a sort of opposite (join) operation with respects to the mining tasks. In effect, this task is used to build a table using previously mined patterns currently stored in a model object. More specifically, since a model is a specification of a set of properties holding in the data, applying a model to data essentially means making such properties explicit in extensional form. In order to be *applied* to data, each data mining model has to implement a decidable containment operator. This task is of paramount importance, since it is capable of putting the mined *knowledge* stored in the model-world back into data-world, for example associating each tuple with the most likely target class according to a given model, or enumerating all frequent patterns appearing within a tuple.



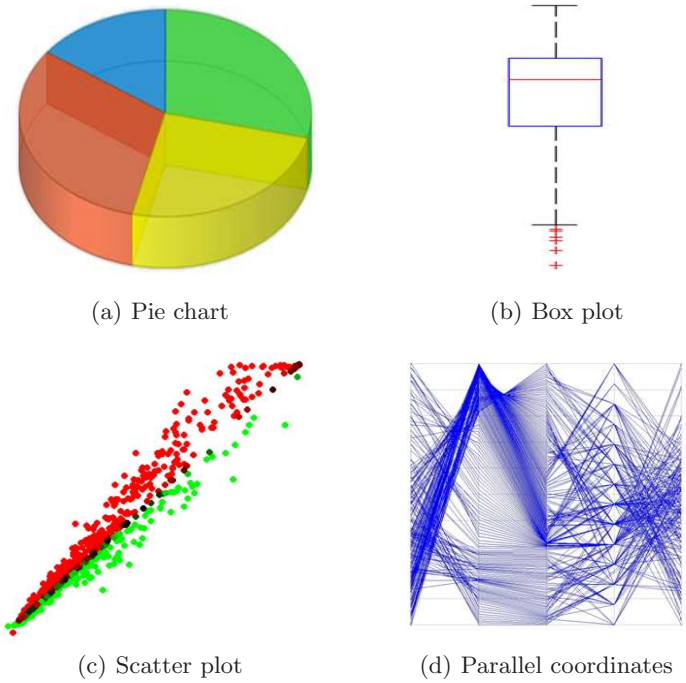
(a) Table export (b) Model export

Fig. 5.11. Export tasks

Figure 5.11 shows a table export task (see 5.11(a)) and a model export task (see 5.11(b)). Those kinds of tasks have one input of type table and model respectively, and no output, since their effect is to store data or models in some external repository.

5.4 Entering tables and models worlds: visualization and statistics

By double clicking on table/model nodes in the workflow perspective, a data/model perspective will be opened respectively. From that perspectives, it is possible to perform data/models analyses by means of graphical exploration and different representations of data/models. Picture on figure 5.3 shows data and model perspectives.



(a) Pie chart (b) Box plot

(c) Scatter plot (d) Parallel coordinates

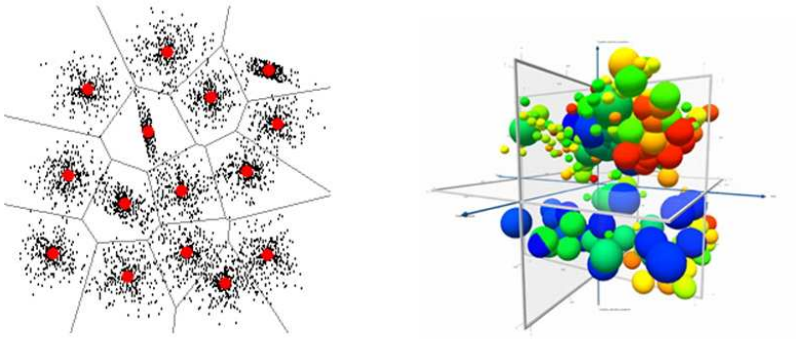
Fig. 5.12. Data visualization

When entering the data perspective, user can specify which statistics or graphical representations to use. Like in the workflow perspective, where a node repository contains all available nodes, in the data perspective there is

a repository where data visualizers are organized into several categories, as specified in the following schema:

- Statistics: contains descriptive statistics divided in several subcategories:
  - *basic*: such as missing values, distinct values, unique values, etc...
  - *numeric*: such as maximum, minimum, mean, standard deviation, median, percentiles, etc...
  - *nominal*: such as mode, value-counter, etc...
- Charts: contains several subcategories:
  - *univariate*:
    - *numeric*: such as histograms, lines, box plots (see 5.12(b)), etc...
    - *nominal*: such as pie charts (see 5.12(a)), bar charts, etc...
  - *multivariate*:
    - *numeric*: such as scatter plots (see 5.12(c)), parallel coordinates (see 5.12(d)), correlation matrixes, etc...
    - *nominal*: such as segment viewers, confusion matrixes, etc...

When entering the model perspective, the user can graphically analyze generated models. As in data perspective, there is a repository where model visualizers are organized into several categories. A main difference between model and data perspective is that model visualizers cannot be applied to all kinds of models. As a matter of fact, models are significantly different from each other (see fig. 4.3), and only specific visualizers are suitable to represent a particular model. Figure 5.13 presents two kinds of model visualizers.



(a) Clustering regions (b) Cluster bubbles

Fig. 5.13. Model visualization

Part III

The **RecBoost** application

6

Boosting text segmentation via progressive classification

6.1 Introduction and motivation

The wide exploitation of new techniques and systems for generating, collecting and storing data has made available a huge amount of information. Large quantities of such data are stored as continuous text. 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. For instance, personal demographic information typically comprises names, addresses, zip codes and place names, which indicate a convenient organization for the these kind of data. However, the extraction of structure from textual data poses several challenging issues, since free text does not necessarily exhibit a uniform representation.

Foremost, the order of appearance of the attributes across the individual lines of text may not be fixed. In addition, their recognition is further complicated by the absence of both suitable field separators and a canonical encoding format, which is mainly due to erroneous data-entry, misspelled terms, transposition oversights, inconsistent data collection a nd so forth [51]. As a concrete example, common issues in personal demographic data are the adoption of abbreviations for both proper names and common terms and the availability of multiple schemes for formatting addresses, phone numbers and birth dates. Also, distinct records may lack different attribute values, which makes them appear with a variable structure. Yet, the same data may be fragmented over disparate data sources, which further exacerbates the aforementioned difficulties.

The notion of *Entity Resolution* [9, 26, 95], denotes a complex process for database manipulation that embraces three primary tasks. *Schema reconciliation* consists in the identification of a common field structure for the information in a data source. *Data reconciliation* is the act of discovering synonymies in the data, i.e. apparently different records that, as a matter of fact, refer to a same real-world entity. *Identity definition* groups tuples previ-

ously discovered as synonymies, and extracts a representative tuple for each discovered group.

In this part of the thesis a novel approach to schema reconciliation, called RecBoost, is proposed, that adopts classification as an effective mechanism for fragmenting free text into tuples with a common attributes structure. 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 such a token with a label denoting its ontological or syntactic category. Reconciliation is 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 here is thus a 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 this 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.

Moreover, due to the variable number of classification stages, RecBoost gives the user finer control over the trade-off between accuracy (i.e. the proportion of correctly classified tokens w.r.t. the classification behavior of the overall RecBoost system) and recall (i.e. the proportion of correctly classified tokens w.r.t. the actual tokens to reconcile). In practice, the user can choose a classifier with a trade-off satisfying the requirements of the specific application.

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. In addition, the combination of rule-based classification models with domain-specific ontologies makes the generic RecBoost classifier very intuitive, i.e. easier to interpret than state-of-the-art probabilistic methods, such as DATAMOLD [12] and Mallet [69].

In order to take advantage from results obtained in the previous stages, we effectively exploit the 2W Model framework defined in 5. Notice that the afore-

mentioned approach can be modeled outside the 2W Model, but the exploitation of our KD framework ease the definition of the overall KD workflow. In the following sections, both the *ad-hoc* modeling and the 2W Model approach will be discussed, showing the benefits of the second mentioned approach.

6.2 Related work

Text reconciliation is clearly related with *Part Of Speech* (POS) *Tagging* and *Shallow Parsing*, *Wrapping* and, in general, with the problem of extracting structure from free text. The aim of POS Tagging is to assign labels to speech words that reflect their syntactic category. To this purpose, both statistical and rule-based techniques [14, 62, 67, 68] have been proposed in the literature. In practice, the basic idea behind POS tagging consists in disambiguating phrases by exploiting dictionaries and analyzing the textual context surrounding each candidate entity. However, the approach fails at treating *exceptions*, i.e., words that are not included in a dictionary, such as proper names, cities, or addresses. By contrast, these are exactly the features which characterize our scenario.

The point is that POS Tagging always allows the assignment of a set of labels to each word and, consequently, focuses on finding the most adequate candidate. However, a term can generally assume very different meanings, depending on both its position in the free text and its neighboring words. Hence, this imposes a different treatment for known lexicons and for the aforementioned exceptions. The former can be labeled by resorting to suitable look-up tables, whereas the latter would require some ad hoc processing, that takes into account the assigned labels. Viewed in this respect, our approach first employs dictionaries for associating reliable labels to common terms in a free text. Then, automatically-generated rules are exploited for classifying those remaining terms that cannot be found in the cited dictionaries.

As far as wrapping is concerned, most algorithms considerably rely on HTML separator tags, and on the fact that data represent a regular multi-attribute list [38]. Such approaches 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 exception are capable of effectively dealing with such features [3, 19, 85]. For example, WHISK [85] 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.

ILP provides a solid framework for designing rule-based systems aimed at text categorization and information extraction [27, 41]. In particular, a divide-and-conquer approach to the problem of learning rules for accomplishing both

these latter tasks is proposed in [60]. In principle, such a technique can be employed for text reconciliation, since it allows the extraction of focussed textual fragments and their subsequent labeling. However, its exploitation for practical applications is problematic, due to the fact that rules have to explicitly locate fragment boundaries. This imposes a non-trivial learning phase, that requires two distinct sets of training examples, respectively necessary for denoting what specific (aggregates of) words can be taken into account as possible boundaries within the underlying textual data and for specifying how to label their intermediate fragments. Also, fragment extraction relies on tests on the occurrences of domain-specific words, that are either learnt from the training examples, or exhibit some degree of positive correlation to such examples. However, locating all relevant fragments determined by meaningful combinations of these words is computationally unfeasible. This imposes the exploitation of various indexing structures to accelerate the evaluation of rule predicates on the underlying text. By contrast, RecBoost pursues token-by-token reconciliation, thus overcoming all of the above issues related to fragment boundaries.

Several recent approaches to schema reconciliation rely on Hidden Markov Models (HMM) [4, 12, 64, 42, 76]. Schema reconciliation with HMM can be accomplished by learning the structure of a HMM, and applying the discovered structure to unknown examples. As an example, DATAMOLD [12] employs a training phase to learn a HMM, that consists of a set of states and directed edges among such states. Two particular states are the *initial* and the *final* states. The former has no incoming edges, whereas the latter has no outgoing edges. Every state of the HMM, except from the *initial* and *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, DATAMOLD pursuits classification 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. Indeed, in order to associate terms that do not appear in the training set with a corresponding state, DATAMOLD relies on *smoothing*, i.e. on the exploitation of ad-hoc probabilistic strategies.

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, when-

ever 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. This latter aspect is clarified in fig. 6.1, where it is shown that the HMM topology prevents the correct reconciliation of the input sequence Harry Hacker London. Notice that node labels indicate input tokens for the automaton. Moreover, the color of node labels corresponds to actual token classes (i.e. name/green, surname/red and city/blue), whereas the color of a node contour denotes the class assigned by that node to its label. Clearly, discrepancies between node and label colors represent reconciliation errors. The illustration shows that four paths exist in the automaton and, hence, as many alternatives to reconcile the input sequence. However, all such paths lead to erroneous reconciliations. The only sequence of states in the automaton, that would lead to a correct reconciliation, does not form a path between the ending states $I.S.$ and $F.S.$, thus being unemployable to reconciliation purposes.

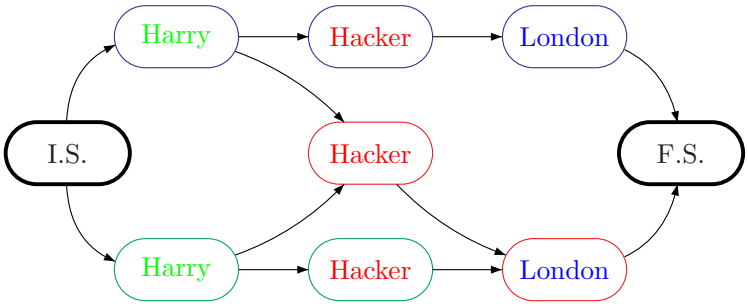


Fig. 6.1. Unreliable token reconciliation due to HMM topology

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. Consider the sequence Harry 348.2598781 London, where the second token represents a phone number. Although the correct label is unknown to the model shown in fig. 6.1, the latter will still try to assign a known label to the token. As a consequence, the whole sequence will result in a low fitting. It is worth noticing, however, that some regularities in the structure (e.g. the high probability that a sequence starts with a name) would allow to correctly classify the "known part" of the sequence, by avoiding to classify the "unlikely" term.

Recently, emphasis has 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, Maximum Entropy Markov Models (MEMMs) [42], 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 contex-

tualizing token reconciliation. However, MEMMs suffer from the well known label-bias problem [64].

Conditional random fields (CRFs) [64, 69] are a probabilistic framework, that can be employed for text labeling and segmentation. 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. Still, such improvements in the HMM technology 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.

Though CRFs outperform both MEMMs and HMMs in a variety of applicative scenarios, including bioinformatics, computational linguistics and speech recognition,

An unsupervised approach to text reconciliation is introduced in [4]. 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. The approach consists of a two-step process. Assume that R is 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. Initially, a preprocessing step 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.

A fixed three-layered topology and the exploitation of a feature hierarchy, i.e. a set of generalizations for basic values, make each *ARM* capable of dealing with unknown tokens and robust to noise within input data. A fixed BMT topology is adopted for *ARMs*. This guarantees accurate text segmentation and robustness to noise in the reference table, while avoiding the reliance on cross-validation techniques and stochastic optimizations [12] to automatically decide proper HMM topologies, that would otherwise require several scans of the underlying reference table.

The *ARMs* of all attributes can be exploited to determine the best segmentation of an input string at the second (run-time) step. This involves to first learn the total order of attributes from a batch of input strings and to subsequently 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_{s_i} maximizes the overall reconciliation quality $\prod_{i=1}^{n} ARM_{s_i}(s_i)$ among all possible segmentations.

Notice that the exploitation of reference tables is a natural way of automatically building training sets for the text 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 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.

As to a more specific comparison with our contribution, the reference table approach [4] 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.

Furthermore, the reference table approach adopts ARMs for reconciliating individual attribute values. However, ARMs are basically HMMs and, hence, suffer from the 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.

6.3 Notation and preliminaries

To formalize the Schema Reconciliation problem, we assume the basic definitions in the following section.

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) $S = \{s_1, \ldots, s_m\} \subseteq \mathcal{M}^*$, we want to segment each sequence s_i into subsequences $s_i^1, \ldots, s_i^n$, such that each token $a \in s_i^h$ is associated with the proper attribute A_j.

For example 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 fig. 6.2.

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. 6.2. (a) Unstructured data. (b) Reconciled data.

Text reconciliation can be profitably employed in several contexts. We briefly mention a wide range of major applications, to provide a taste of the generality of our approach. The harmonization of postal addresses affects large organizations like banks, telephone companies and universities, which typically collect millions of unformatted address records. Since each address-record can, in principle, be retrieved from a different data source (designed with different purposes), variations in the way such records are stored are far

from unusual. Further applicative scenarios requiring to deal with the schema reconciliation problem include processing bibliographic records, collections of information about products, medical sheets, and so forth.

A typical applicative setting which motivates our work is the mentioned entity resolution problem, which roughly consists in discovering/disambiguating duplicate tuples [26, 9, 85]. When tuples consist of free text, which however contains a hidden structure, tuple disambiguation can be accomplished by exploiting either exact matching techniques, based on specific segments of the strings, or simpler (fuzzy) techniques, that ignore segmentation. 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

that 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

In principle, various schemes may be followed to catch string resemblance. Without loss of generality, we here focus on two basic approaches, namely Jaccard similarity and weighted Jaccard similarity. The former measure catches resemblance between any two strings by measuring their degree of overlap, i.e. the proportion of common tokens. Formally, the Jaccard similarity between the above strings s_1 and s_2 is defined as

$$d_J(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}$$

The latter measure weights segment matches by the relevance of the corresponding attributes. Let w_1, w_2, w_3, w_4 be four weights respectively denoting the relevance of segment matches in correspondence of the attributes NAME, ADDRESS, CITY and STATE. The definition of the weighted Jaccard similarity between the foresaid strings s_1 and s_2 is

$$d'_J(s_1, s_2) = \frac{\sum_{i=1}^{4} w_i d_J(a_i, b_i)}{\sum_{i=1}^{4} w_i}$$

where entities a_i and b_i denote two corresponding segments, respectively within s_1 and s_2.

It is easy to see that $d_J(s_1, s_2) = 0.44$. Such a result does not fully reflect the evident similarity of s_1 and s_2, which may generally prevent their

disambiguation. To overcome such a limitation, it is sufficient to reasonably assume that a low degree of overlap between string segments corresponding to the ADDRESS and STATE attributes should not heavily penalize the overall similarity. Indeed, multiple addresses can be associated to an individual in a same city, whereas the latter resides in only one state. By accordingly fixing $w_1 = 0.5$,$w_2 = 0.1$,$w_3 = 0.35$ and $w_4 = 0.05$, it follows that $d'_J(s_1, s_2) = 0.71$, which more appropriately reflects the actual resemblance between the two strings under comparison. This confirms that exact matching techniques enable more effective string de-duplication, whenever the original strings can be accurately segmented.

Thus, a de-duplication system adopting a text segmentation methodology, would effectively leverage its performance, provided that the embedded segmentation methodology is reliable.

Reliability here has a strict meaning: strings should be correctly segmented, and errors in segmenting should be absolutely avoided. Indeed, a wrong segmentation would likely result in a worsening of the de-duplication effectiveness, even when compared to simpler schemes. As an example, let us consider the following wrong segmentation of the above strings:

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

In such a case, it still holds that $d_J(s_1, s_2) = 0.44$, whereas $d'_J(s_1, s_2) = 0.125$. In other words, though previously shown more effective, weighted Jaccard similarity now reveals unreliable. Instead, simpler schemes disregarding segmentation, such as basic Jaccard similarity, could still somehow enable string disambiguation in an acceptable way.

The point is the trade-off between precision (i.e., the capability of correctly segmenting a tuple) and recall (i.e., the capability of segmenting a whole). It is clear that classification systems exhibiting high precision, even at the cost of low recall, can be safely embedded into a deduplication system, according to the scenario described in fig. 6.3.

In this scenario, a text segmentation system has two choices: either a sequence can be safely segmented, and the result of the segmentation is reliable, or the segmentation result is not affordable. In the former case, exact matching techniques based e.g. on weighted Jaccard similarity can be applied, whereas the latter case can still provide a reconciliation, which is however based on fuzzier schemes. Unfortunately, the state-of-the-art approaches from current literature, based on probabilistic modeling, do not properly fit the above scheme, since they foresee a segmentation even in presence of high uncertainty.

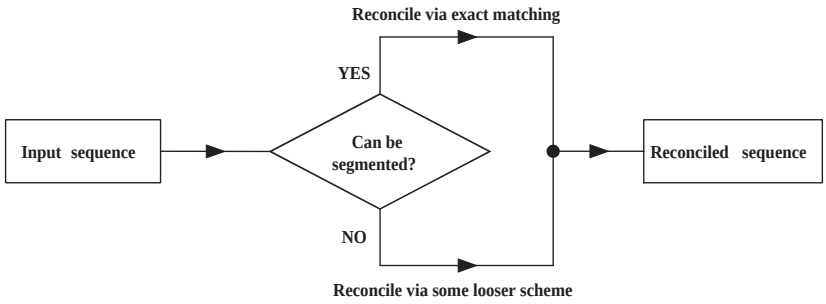


Fig. 6.3. Example reconciliation

RecBoost is 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.

6.4 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. Using complex data type defined in 5.2.1, the aforementioned sequences can be modeled using event collections. More precisely, a sequence s is modeled by an event collection ec whose events $e_1, ..., e_k$ represent token $a_1, ..., a_k$ of the sequence s. Since we are going to show both the *ad-hoc* approach and the KD framework approach, from now on, we use both the terms *sequence* and *event collection* to denote a string that has to be reconciled. Similarly, we use both the terms *token* and *event* to refer to a particular element of the aforementioned string.

RecBoost pursues text reconciliation via term generalization. Precisely, two types of generalizations are involved, namely syntactic and ontological analysis, and contextual generalization. The former aims at labeling 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 the ones employed in [4, 12], to preprocess sequences within S. In practice, a domain ontology is specified as $\mathcal{G} = \langle \mathcal{L}, \triangleleft, \mathcal{A} \rangle$, where $\mathcal{L}$ is a set of categories, $\triangleleft$ 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*

$\mathcal{L}$ 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 $\triangleleft$ relation. Fig. 6.4 shows an exhaustive set of concepts and their hierarchical relationships.

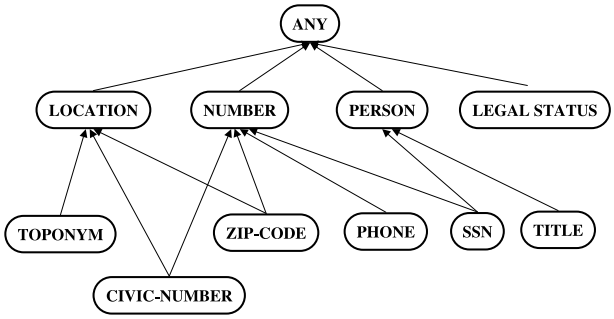


Fig. 6.4. A concept hierarchy for personal information in a banking scenario

$\mathcal{A} = \{r_1, \ldots, r_h\}$ is a set of rules, that 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. Formally, the generic rule $r_i \in \mathcal{A}$ relates a basic textual token with some corresponding ontological concept in $\mathcal{L}$, i.e. $r_i : \mathcal{M}^* \rightarrow \mathcal{L}$. Generally, the prior definition of a number of rules allows to properly deal with several tokens in a wide variety of applicative settings, with no substantial human effort.

As to the interpretation of ontological rules, *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,

exemplified in the context of the ontological rules $\mathcal{A} = \{r_1, r_2, r_3\}$ adopted for the concept hierarchy of fig. 6.4. The following two rules r_1 and r_2 involve both type of actions, in : r_2 *tag* is a context of event e (see refrealizing:eventCollection).

	ad-hoc system	*KD framework*
r_1:	**if** a is a four-digits token **then** **replace** a with ZIP-CODE	**if** e is a four-digits event **then** $e.tag$ = ZIP-CODE

	ad-hoc system	*KD framework*
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}	**if** e_i is a four-digits event **and** e_{i+1} is an event containing digits **then** **merge** e_i and e_{i+1}

In general, *relabeling* 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).

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

	ad-hoc system	*KD framework*
r_3:	**if** $a \in$ DICTIONARY **then** **replace** a with TOPONYM	**if** $e \in$ DICTIONARY **then** $e.tag$ = TOPONYM

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

[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.

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 subsection.

Contextual analysis.

This step is meant to associate tokens (event collections) 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 (event) 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 (event collection) $s \in T$ and for each token (event) $a_i \in s$, the label $\lambda_s(a_i)$ is known.

In order to correctly classify each token $a_i \in s$, we exploit information about its *neighborhood*. More specifically, the neighborhood $feature_s(a_i)$ of a generic token $a_i \in s$, is the set of all the items preceding and following a_i in s and is formally defined as follows:

$$feature_s(a_i) = \langle pre_s(a_i),\ a_i\ , post_s(a_i)\rangle$$

In the above notation, $pre_s(a_i)$ is the fragment of $feature_s(a_i)$ that precedes a_i. Dually, $post_s(a_i)$ indicates the context segment that follows a_i. The set $\mathcal{T} = \{\langle feature_s(a), \lambda_s(a)\rangle | s \in T, a \in s\}$ represents the training set for our classification problem.

Using event collections, instead, we can easily model $pre_s(a_i)$ and $post_s(a_i)$ as event *contexts*. In practice, tokens (event) in $pre_s(a_i)$ will be grouped in an event collection, and such event collection will be the value of the *e.pre* context. Similarly, tokens (event) in $post_s(a_i)$ will be grouped in an event collection, and such event collection will be the value of the *e.post* context.

The idea beyond contextual analysis is to examine the context $feature_s(a)$ 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:

if $Condition$
then $\lambda_s(a) = Class$

Here, a and s represent, respectively, token (events) and sequence (event collection) 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 $a = v$, $v \in pre_s(a)$ or $v \in post_s(a)$, where v is any constant in $\mathcal{M} \cup \mathcal{L} \cup R$. Working with event collections, instead, we will have: $e = v$, $v \in e.pre$ or $v \in e.post$

The latter two conditions strictly relate token reconciliation with context inspection. Indeed, condition $v \in pre_s(a)$ (resp. $v \in post_s(a)$) narrows context analysis to what actually precedes (resp. follows) token a.

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 [5] is exploited to extract from the explicit representation of token contexts, $\mathcal{D} = \{\langle feature_s(a), A\rangle | s \in D, a \in s, A \in R\}$, 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 [53], 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 modeled 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 [53] 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.

6.5 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 [4, 12]. 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 $\mathcal{P} = \{C_1, \ldots, C_k\}$ 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 $\mathcal{P}$ 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 $\lambda_s(a) = Class$

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

	ad-hoc system	*KD framework*
r':	**if** *Condition* **then** **replace** a with $Class$	**if** *Condition* **then** $e.tag = Class$

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 $\langle feature_s(a), \lambda_s(a) \rangle$. Notice that there is a direct correspondence between the context $feature_s(a)$ computed at step i and the context computed at step $i-1$. Indeed, the new context $feature_s(a)$ follows from the context of a within T_{i-1} by replacing each token $b \notin X_s$ of s with its corresponding attribute $C_{i-1}(b)$.

Notice that the aforementioned methodology can be modeled in an easy and elegant way using 2W Model and the relative KD framework, as illustrated in figure 6.5.

On the other hand, outside 2W Model and without event collection data type, the discussed methodology is quite complex to be modeled, and the application has to be supported by three main components, namely a *preprocessor* (*tokenizer*), a *classifier learner* and a *postprocessor*. The components have to cooperate both in the training and in the classification phases, as detailed in figure 6.6. In the following, we explain the role played by each of the aforementioned modules, illustrating also the corresponding proceedings in the KD framework modeling case.

Preprocessor

Initially, a cleaning step has to be 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

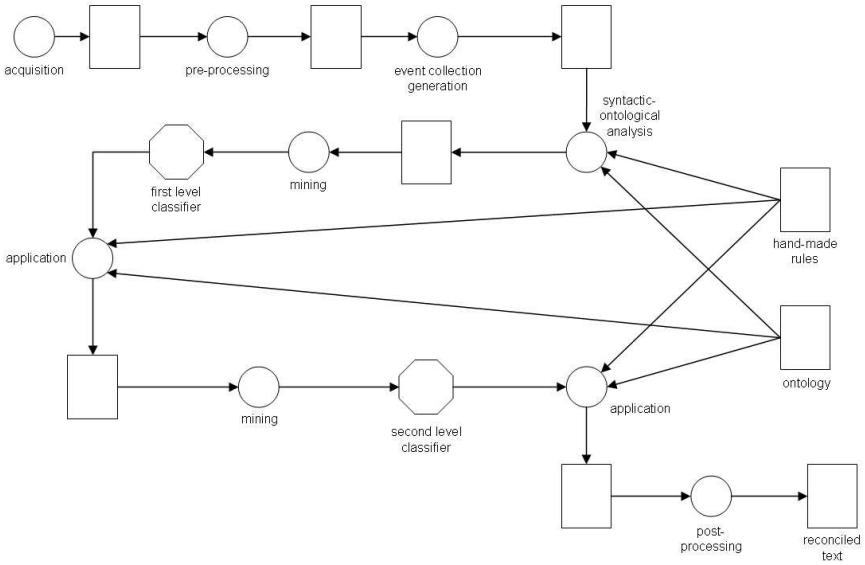


Fig. 6.5. RecBoost methodology in the KD framework

cleaning steps are accomplished by using domain-specific transformation rules suitably defined in $\mathcal{G}$.

In the KD framework, all those operations are performed by task elements (i.e. data filters). In this step, input string are also transformed in event collections objects.

Classifier learner

The *classifier learner* is responsible for producing an optimal set of classification rules, as shown in figure 6.6(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 labeling rules and the $\triangleleft$ 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 labeling 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

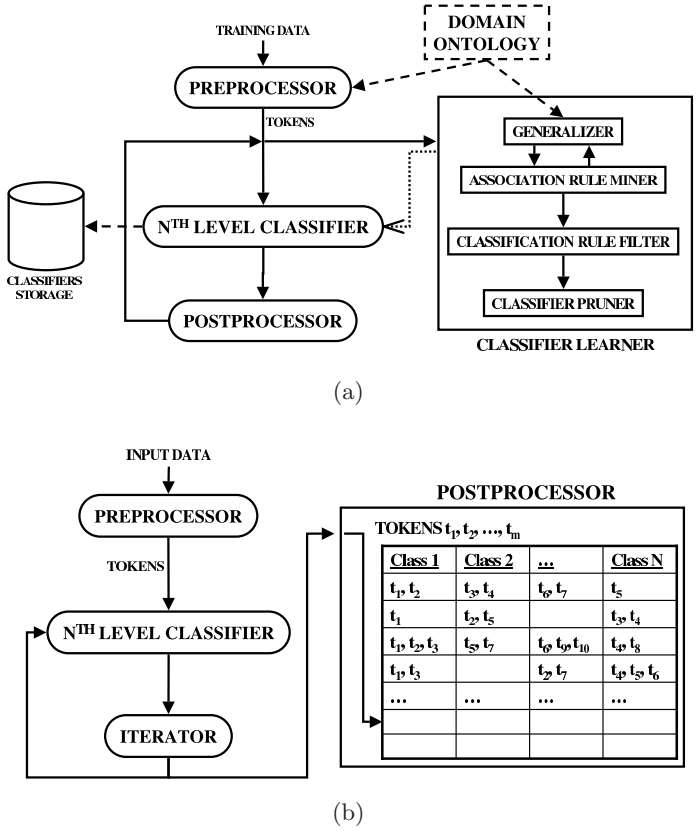


Fig. 6.6. Training (a) and Classification (b) phases in the RecBoost methodology

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.

In the KD framework, all those operations are performed by a mining task in conjunction with a suitable model object.

Post-processor

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 6.6(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 6.6, 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}.

In the KD framework, this step is very simple, since the overall input string structure is practically maintained the same, and the class of a particular event can be directly discovered looking at the *tag* context.

6.6 An illustrative example

Problem formulation

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 containing all known toponyms, and a domain-specific ontology $\mathcal{G} = \langle \mathcal{L}, \lhd, \mathcal{A} \rangle$, where

$$\mathcal{L} = \{\mathsf{PHONE\text{-}NUMBER}, \mathsf{SSN}, \mathsf{TOPONYM}, \mathsf{ZIP\text{-}CODE}\}$$

and $\mathcal{A}$ consists of the ontological rules in fig. 6.7
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.

	ad-hoc system	*KD framework*
r_1:	**if** a is a four-digits token **then** **replace** a with ZIP-CODE	**if** e is a four-digits event **then** $e.tag$ = ZIP-CODE
r_2:	**if** a is a token containing more than four digits **then** **replace** a with PHONE-NUMBER	**if** e is an event composed by more than four digits **then** $e.tag$ = PHONE-NUMBER
r_3:	**if** a is a token of type *ddd-dd-dddd* **and** d is a digit **then** **replace** a with SSN	**if** e is an event of type *ddd-dd-dddd* **and** d is a digit **then** $e.tag$ = SSN
r_4:	**if** $a \in$ DICTIONARY **then** **replace** a with TOPONYM	**if** $e \in$ DICTIONARY **then** $e.tag$ = TOPONYM

Fig. 6.7. Ontological rules

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.

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, in the cases of both ad-hoc system and KD framework.

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

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

As shown in figure above, as far as the first row is concerned, the preprocessor removed two brackets, two inverted commas and a hyphen. In practice, after the preprocessing all sequences are represented with the same format.

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.

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 in fig 6.8.

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.

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,

	ad-hoc system	*KD framework*
r_5:	**if** $pre_s(a) == \emptyset$ **and** *TOPONYM*,*ZIP* $\in post_s(a)$ **then** λ_a = [NAME]	**if** $e.pre == \emptyset$ **and** *TOPONYM*,*ZIP* $\in e.post$ **then** $e.tag$ = [NAME]
r_6:	**if** $a ==$ *TOPONYM* **then** λ_a = [ADDRESS]	**if** $e ==$ *TOPONYM* **then** $e.tag$ = [ADDRESS]
r_7:	**if** *TOPONYM* $\in pre_s(a)$ **and** *ZIP* $\in post_s(a)$ **then** λ_a = [ADDRESS]	**if** *TOPONYM* $\in e.pre$ **and** *ZIP* $\in e.post$ **then** $e.tag$ = [ADDRESS]
r_8:	**if** $a ==$ *ZIP* **then** λ_a = [ZIP]	**if** $e ==$ *ZIP* **then** $e.tag$ = [ZIP]
r_9:	**if** *TOPONYM*,*ZIP* $\in pre_s(a)$ **and** $post_s(a) == \emptyset$ **then** λ_a = [CITY]	**if** *TOPONYM*,*ZIP* $\in e.pre$ **and** $e.post == \emptyset$ **then** $e.tag$ = [CITY]

Fig. 6.8. Classification rules

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

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

Second-level classifier

To this point, as a consequence of progressive classification, the original domain information in $\mathcal{G}$ is updated to yield an enriched ontology $\mathcal{G}' = \langle \mathcal{L}, \triangleleft, \mathcal{A}' \rangle$. The set $\mathcal{A}'$ of ontological rules is obtained by augmenting the original one, $\mathcal{A}$, with the classification rules learnt at the end of the previous reconciliation step. Formally, $\mathcal{A}' = \mathcal{A} \cup \{r_5, r_6, r_7, r_8, r_9\}$.

Without using the KD framework, first-level classification rules have to be transformed into labeling rules before being added to $\mathcal{A}'$. Instead, using the KD framework, there is no need to transform classification rules, since this step is accomplished by an application task.

Contextual analysis is then 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, and it is composed by the following individual rule:

	ad-hoc system	*KD framework*
r_{10}:	**if** [NAME] $\in pre_s(a)$ **and** [ADDRESS],[ZIP] $\in post_s(a)$ **then** λ_a = [NAME]	**if** [NAME] $\in e.pre$ **and** [ADDRESS],[ZIP] $\in e.post$ **then** $e.tag$ = [NAME]

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.

Post-processor

The post-processor rebuilds the original sequence s_1, by fitting its corresponding tokens in a suitable structure defined by the descriptor $R = \{\mathsf{NAME}, \mathsf{ADDRESS}, \mathsf{ZIP}, \mathsf{CITY}\}$. If using the KD framework, this step is accomplished by a task that transforms event collections into a relational structure, basing on events contexts.

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

7

Evaluating RecBoost

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 6.4. 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 6.5. 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.

7.1 Basic setup and performance measures definitions

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 consisted of 24,000 sequences, with an average of 8 tokens per sequence. The schema to reconcile consisted of the fields *Name*, *Address*, *Zip*, *State/Province*, and *City*.
- `BigBook`, a publicly-available dataset[1] consisting of a set of business addresses. Each business description consists of the 6 items *Name*, *Address*,

[1] http://www.isi.edu/info-agents/RISE/repository.html

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[2]. 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 fig. 6.4. 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 sec. 6.6 (as relabeling rules) and rule r_2 at sec. 6.4 (as a restructuring rule).

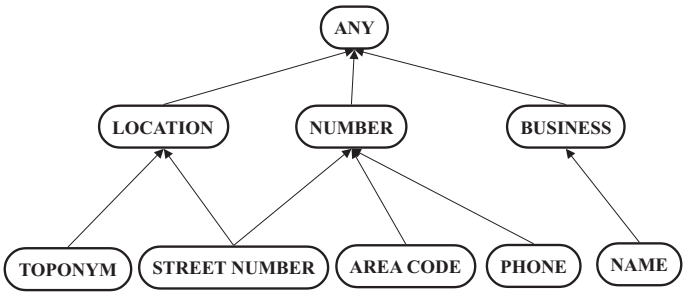


Fig. 7.1. The concept hierarchy for the `BigBook` dataset

The ontology employed for the `BigBook` dataset embraces 9 concepts and is shown in fig. 7.1.

In such a context, no use is made of restructuring actions, so that background knowledge reduces to the relabeling rules shown in fig. 7.2.

Finally, the ontology for the `dblp` dataset is shown in fig. 7.3. Again, background knowledge only involves relabeling rules, that are reported in fig. 7.4.

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*

[2] http://www.informatik.uni-trier.de/ ley/db/

	ad-hoc system	*KD framework*
r_1:	**if** $a \in$ DICTIONARY **then** **replace** a with TOPONYM	**if** $e \in$ DICTIONARY **then** $e.tag =$ TOPONYM
r_2:	**if** a is a token of type *ddd-dddd* **then** **replace** a with PHONE NUMBER	**if** a is an event of type *ddd-dddd* **then** $e.tag =$ PHONE NUMBER
r_3:	**if** a is a token of type *ddd* **then** **replace** a with AREA CODE	**if** a is an event of type *ddd* **then** $e.tag =$ 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	**if** a is a digit-sequence followed by *TH* or *ST* or *ND* or *RD* **then** $e.tag =$ STREET NUMBER
r_5:	**if** a is a digit-sequence **then** **replace** a with NUMBER	**if** e is a digit-sequence **then** $e.tag =$ NUMBER

Fig. 7.2. Relabeling rules

and *via*. DELIMITER DICTIONARY is a set of token delimiters, that comprises *'*, *"*, *;*, *,*, *-*, *.*, and ***.

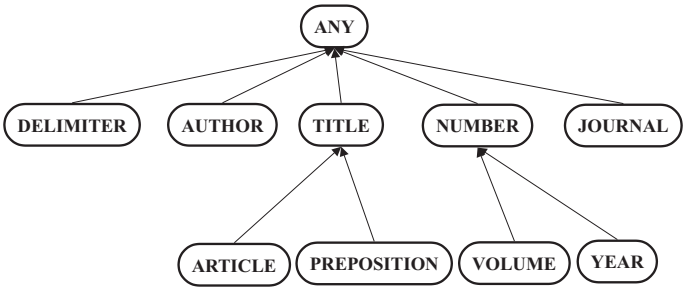


Fig. 7.3. The concept hierarchy for the `dblp` dataset

It is worth noticing that the analysis of the above domain-specific ontologies reveals a key feature of RecBoost methodology. Roughly speaking, it can

	ad-hoc system	*KD framework*
r_1:	**if** $a \in$ Journal Dict. **then** **replace** a with JOURNAL	**if** $e \in$ Journal Dict. **then** $e.tag =$ JOURNAL
r_2:	**if** a is a four-digits token **and** $a \in [1950, 2006]$ **then** **replace** a with YEAR	**if** e is composed by four-digits **and** $e \in [1950, 2006]$ **then** $e.tag =$ YEAR
r_3:	**if** a is a digit-sequence **then** **replace** a with NUMBER	**if** e is a digit-sequence **then** $e.tag =$ NUMBER
r_4:	**if** $a \in$ Delimiter Dict. **then** **replace** a with DELIMITER	**if** $e \in$ Delimiter Dict. **then** $e.tag =$ DELIMITER
r_5:	**if** $a \in$ Gramm-Art Dict. **then** **replace** a with ARTICLE	**if** $e \in$ Gramm-Art Dict. **then** $e.tag =$ ARTICLE
r_6:	**if** $a \in$ Preposition Dict. **then** **replace** a with PREPOSITION	**if** $e \in$ Preposition Dict. **then** $e.tag =$ PREPOSITION

Fig. 7.4. Relabeling rules

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, that 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 $\mathcal{F}$ measure 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, such as the one described in section 6.3. 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 $(1, 10]$.

7.2 Experimental evaluation

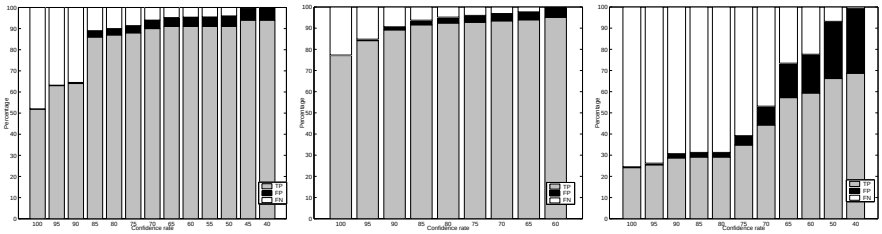
7.2.1 Basic classifier system

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 7.5 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`) 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.

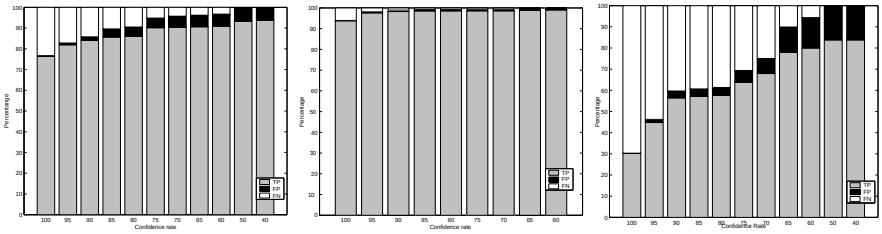
The `dblp` dataset is particularly interesting to investigate in this context, since it exhibits the worst performances. The best we can obtain in this dataset is with confidence set to 40%, which guarantees a significantly high percentage (30.52%) of misclassified tokens. A "safer" confidence value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



(a) `Addresses` (b) `BigBook` (c) `dblp`

Fig. 7.5. Classification results, single stage of classification

We now present several charts in order to discuss upon system accuracy, all those charts can be easily generated using the data perspective (see 5.4) of the KD framework. Figure 7.6 describes the accuracy of the classifier with the adoption of domain-specific concept hierarchies. We exploited the hierarchies described in figures 6.4, 7.1 and 7.3 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.



(a) `Addresses` (b) `BigBook` (c) `dblp`

Fig. 7.6. Classification results with the exploitation of concept hierarchy

Results in the above figures were obtained by exploiting the pruning steps detailed in section 6.4. Indeed, the contribution of the classifier pruner to the misclassification rate is investigated in table 7.1, 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 7.1. Pruning effectiveness

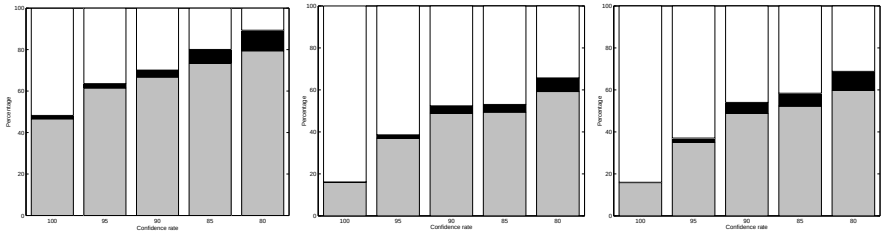
7.2.2 Multiple classification stages

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 7.5 and 7.6);
- 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 7.7 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 fig. 7.6). Again, support was set to 0.5% and confidence was ranged between 100% and 80%.

As shown in the figure, the second-level classifier is in general able to correctly classify a portion of the data, that were unlabeled 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 7.2 summarizes the cumulative results achieved by two levels of classification over the employed datasets.



(a) `Addresses` (b) `BigBook` (c) `dblp`

Fig. 7.7. Classification results at the second stage of classification.

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

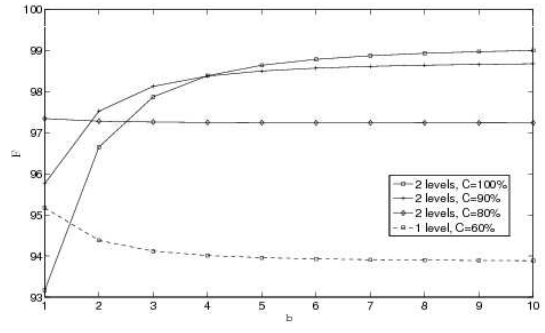
The effectiveness of the second stage of classification is even more evident in the graphs of fig. 7.8. The graphs depict the trend of $\mathcal{F}$ for different values of β. The graphs compare a selection of 2-level classifiers with the single-level classifier (among those shown in fig. 7.6) exhibiting 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:

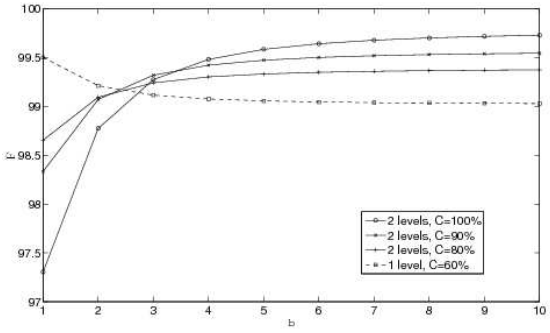
1. how many levels allow to achieve an adequate performance?
2. how should the parameters at each level be tuned?

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 7.9(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.

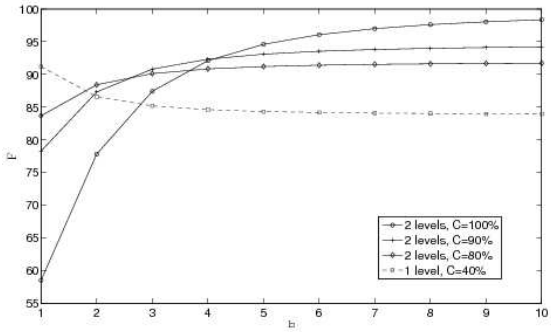
As we can see from figure 7.9(b), however, this strategy does not necessarily work: although the number of misclassified tokens is kept low, the capability of



(a) `Addresses`



(b) `BigBook`



(c) `dblp`

Fig. 7.8. Trends of $\mathcal{F}$-measures compared.

each classifier to recover tokens unclassified in the previous stages decreases. The 5th level looses the capability to further classify tokens, thus ending defacto the classification procedure. Figure 7.9(a) shows the cumulative results at each level.

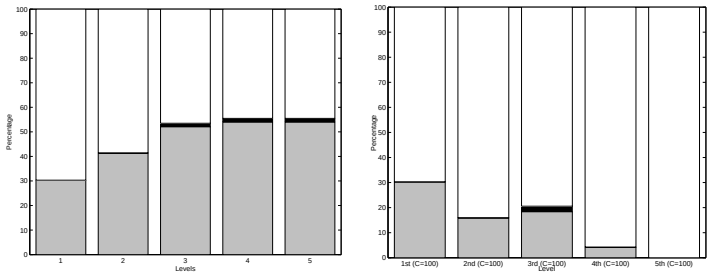
Thus, an upper bound in the number of stages can be set by the classification capability of the stages themselves. A smarter tuning of the parameters which rule the performance of each single stage, allows to achieve best classification accuracy. Figures 7.9(c) and (d) report a different classifier, 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 7.9(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 fig.7.9(e), for example, each classification stage should classify at least 20% of the available tokens, and should misclassify at most 5% of them. Figure 7.9(g), 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 7.9(h) describes the tuning occurred at each stage.

Similar conclusions can be drawn with the other datasets: fig. 7.10, 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.

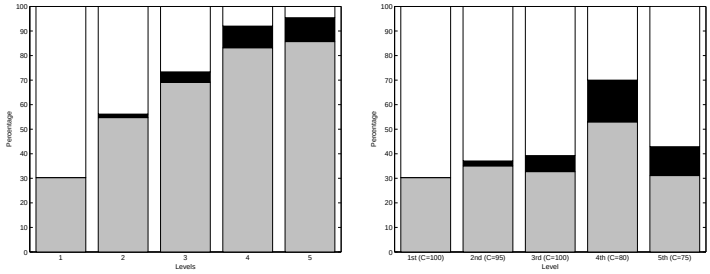
The adoption of multiple classification stages over `BigBook` deserves further discussion about the relation between the size of the labeled data and the number of classification levels which can be defined. Each classification level should build on a separate training set (preprocessed by the preceding 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 learning 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,



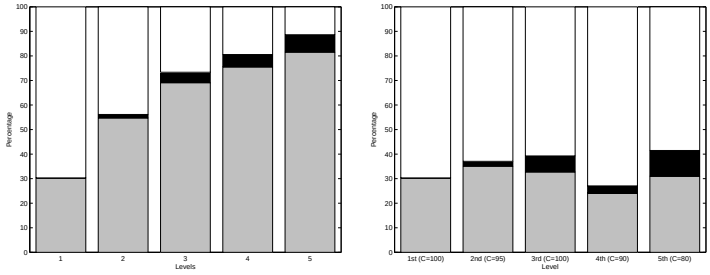
(a) 5 levels (cumulative)

(b) 5 levels (stages)



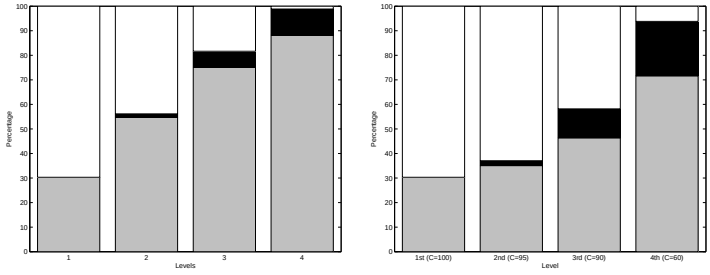
(c) 5 levels (cumulative)

(d) 5 levels (stages)



(e) 5 levels (cumulative)

(f) 5 levels (stages)

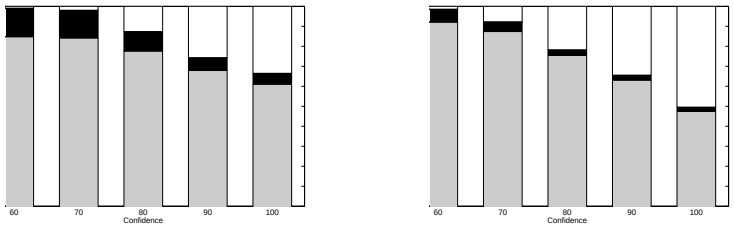


(g) 4 levels (cumulative)

(h) 4 levels (stages)

Fig. 7.9. Effects of multiple classification levels on `dblp`

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.

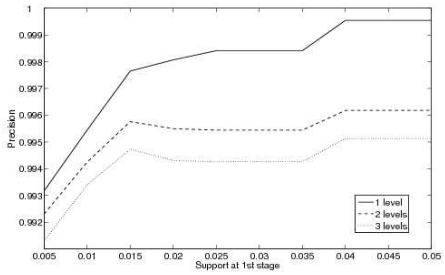


(a) `Addresses` (b) `BigBook`

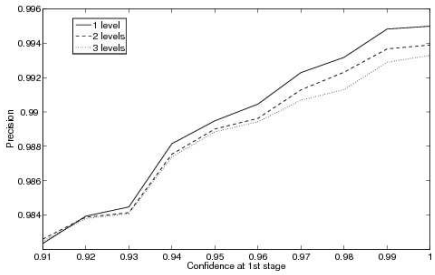
Fig. 7.10. Effects of multiple classification levels.

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 above figures 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` dataset. 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 7.11(c) and 7.11(a) confidence is fixed to 98% and support varies, whereas in figures 7.11(d) and 7.11(b) support is set to 0.5% and confidence varies. Figures 7.11(a) and 7.11(c) show that classification accuracy and recall do not significantly change, especially at higher levels. By the converse, even small variations in the confidence cause significant changes, as testified by figures 7.11(b) and 7.11(d).

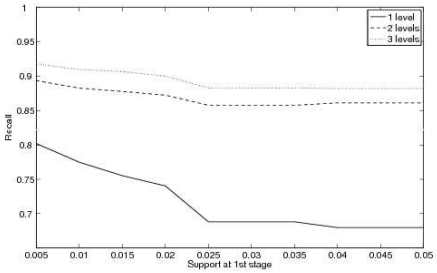
It is interesting to see that, in the above described experiments, the average number of rules which are exploited is nearly stable even on different values of support and confidence (which instead affect the number of discovered rules). Figures 7.12(a) and 7.12(b) depict such a situation. In general, 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



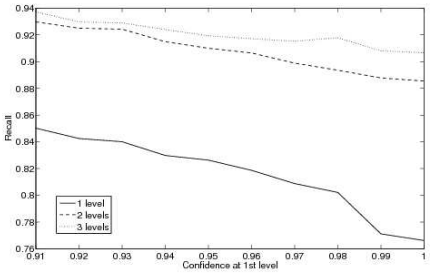
(a) Precision vs. support threshold



(b) Precision vs. confidence threshold

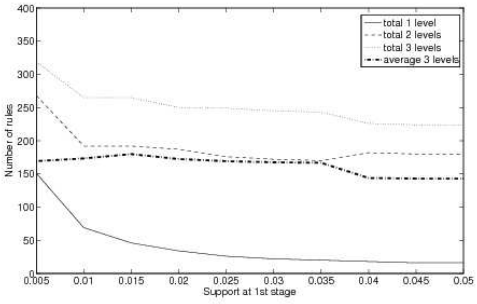


(c) Recall vs. support threshold

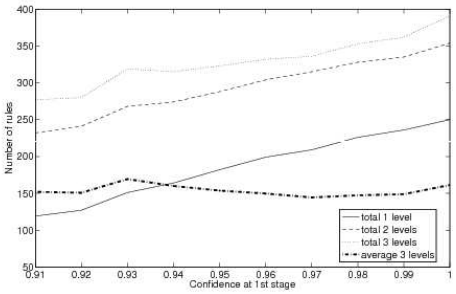


(d) Recall vs. confidence threshold

Fig. 7.11. Stability of Precision and Recall with variable parameter values



(a) Number of rules vs. support



(b) Number of rules vs. confidence

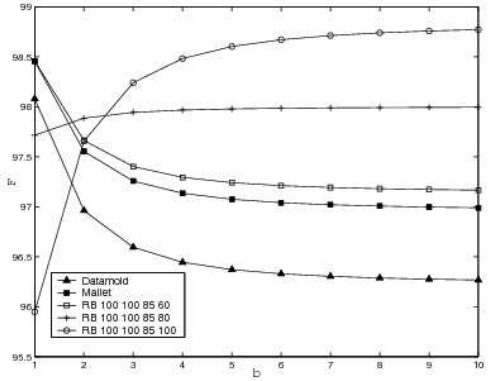
Fig. 7.12. Size of classifiers and average number of rules applied

number of rules applied fluctuates around 50% of the total number of rules obtained in correspondence of the maximum values of support and confidence.

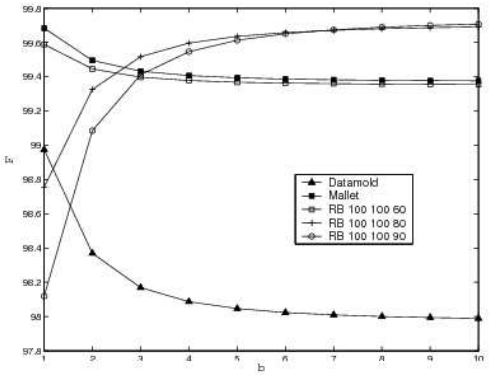
7.2.3 Comparative analysis

The exploitation of the "recursive boosting" strategy proposed in this paper 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 end 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.

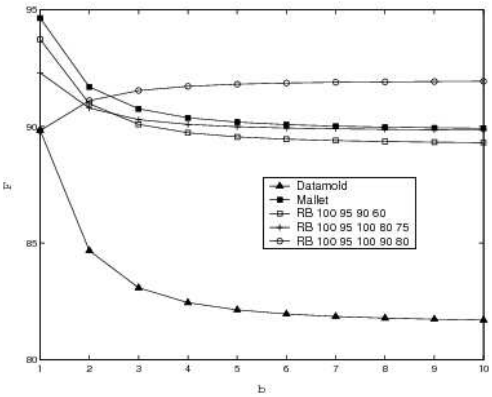
In the following we provide a comparison by exploiting the datasets described in the previous sections. We compare our system with the Mal-



(a) `Addresses`



(b) `BigBook`



(c) `dblp`

Fig. 7.13. Trends of $\mathcal{F}$-measures compared.

let system [69], which provides the implementation of Conditional Random Fields [64] and with the DataMold system [12]. We refer the reader to section 6.2 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 CRF implemented by Mallet was provided by resorting to the Pre/Post information.

An overall comparison is shown in the graphs of fig. 7.2.3, which plot the $\mathcal{F}$ values obtained by Mallet, DataMold, and several different instantiations of the RecBoost system. In particular, we consider the classifiers of figures 7.9 and 7.10, 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 possibility of maintaining an acceptable value of TP. For the `dblp` dataset, we also show an instantiation

As we can see from the figure, the gain in the $\mathcal{F}$ value is evident for $\beta > 2$. Table 7.3 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 fig. 7.6. 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, 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.

Finally, two major arguments emerge in favor of RecBoost as a further result of our comparative analysis.

- Due to the variable number of classification stages, RecBoost gives the user better control over the trade-off between accuracy and recall. In practice, the user can choose a classifier with a trade-off satisfying the requirements of the specific application.
- The generic RecBoost classifier is easier to interpret than existing methods such as DATAMOLD [12] and Mallet [69], since it produces symbolic rules using vocabulary from a domain-specific ontology.

Methods	Addresses							
	TP	FP	FN	P	R	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$
DataMold	96.23%	3.77%	0%	96.23%	100%	98.08%	96.96%	96.59%
Mallet	96.96%	3.04%	0%	96.96%	100%	98.45%	97.55%	97.25%
RecBoost^{1}	93,74%	6,24%	0,02%	93,76%	99,98%	96.77%	94.94%	94.34%
RecBoost$^{\square}$	96,96%	2,86%	0,18%	97,14%	99,81	98.45%	97.66%	97.40%
RecBoost^{+}	95,53%	1,95%	2,52%	98,00%	97,43	97.71%	97.88%	97.94%
RecBoost$^{\bigcirc}$	92,21%	1,09%	6,70%	98,83%	93,23%	95.95%	97.65%	98.24%

Methods	BigBook							
	TP	FP	FN	P	R	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$
DataMold	97.97%	2.03%	0%	97.97%	100%	98.97%	98.36%	98.16%
Mallet	99,37%	0,63%	0%	99,37%	100%	99.68%	99.49%	99.43%
RecBoost^{1}	99,01%	0,98%	0,01%	99,02%	99,99%	99.50%	99.21%	99.11%
RecBoost$^{\square}$	99,21%	0,65%	0,14%	99,35%	99,83%	99.59%	99.44%	99.40%
RecBoost^{+}	97.55%	0.28%	2.17%	99.71%	97.82%	98.75%	99.32%	99.51%
RecBoost$^{\bigcirc}$	96,31%	0,25%	3,44%	99,74%	96,55%	98.11%	99.08%	99.41%

Methods	dblp							
	TP	FP	FN	P	R	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$
DataMold	81,55%	18,45%	0%	81,55%	100%	89,83%	84,67%	83,08%
Mallet	89,83%	10,17%	0%	89,83%	100%	94,64%	91,69%	90,75%
RecBoost^{1}	83,80%	16,20%	0%	83,80%	100%	91,18%	86,60%	85,18%
RecBoost$^{\square}$	88,20%	10,66%	1,14%	89,22%	98,73%	93,73%	90,97%	90,09%
RecBoost^{+}	85,69%	9,74%	4,57%	89,80%	94,94%	92,30%	90,78%	90,29%
RecBoost$^{\bigcirc}$	81,53%	7,10%	11,37%	91,98%	87,76%	89,82%	91,10%	91,54%

Table 7.3. Comparison against Mallet and DataMold.

7.3 A case study

7.3.1 Risk analysis in a bank intelligence scenario

In this section we propose a hash-based technique for data reconciliation, i.e. the recognition of apparently different records that, as a matter of fact, refer to the same real-world entity.

Risk analysis is a key business analysis task for bank organizations, since these typically invest considerable amount of money to support customers' business initiatives. Precisely, risk analysis is required to identify and assess those critical factors that may negatively affect customer successful payback or somehow lower the expected revenues.

In our scenario, the starting point is the construction of a specific data mart capable of ranking the credit risk of a customer by looking at the past insolvency history involving the customer. Usually, a business transaction identifies a creditor and a debtor. A bank agency acts as a mediator in the transaction, by supporting the creditor in collecting the amount she claims. This is accomplished by providing billing services, or by anticipating the monetary flow. Here, risk analysis is accomplished by identifying all possible critical factors for a given transaction. Next, the occurrence probabilities of such critical factors are computed. These probabilities are eventually employed both to identify suitable measures that help prevent critical factors from occurring and to

develop effective risk management plans, i.e. countermeasures that allow the bank organization to avoid/recover from insolvencies, partial paybacks and unexpectedly low revenues.

Risk assessment for a given transaction can greatly benefit from the advantages offered by the data warehousing technology. Among these, fast data access and predictive multidimensional analysis. In particular, the latter enables and expedites a number of management and/or analytical tasks. The multidimensional data warehouse design allows the user (either executive, manager, analyst) to perform very complex business analysis over million of transactions in an easy and effective manner. Examples of typical queries are:

- how does the transaction counter value correlate with the quarterly unsolved effects?
- which is the volume of effects regularly met per district and working categories during last year?
- how does effect collection carry on during the year?

However, the definition of such a data warehouse in a real-life operational scenario is quite complex, and clashes with the impedance mismatch: the same informative unit can be represented into several different ways. This problem is due to the fact that bank data is originally stored within heterogeneous sources, namely separated operational databases, unstructured repositories, legacy systems and papery archives. Moreover, bank data are inherently heterogeneous in nature. Indeed, the foresaid sources of- ten contain data of varying quality and also adopt heterogeneous as well as inconsistent representations, values and formats, that require to be suitably consolidated. In particular, data within legacy systems are not directly accessible and are, hence, typically gathered via reporting functionalities, that conform to proprietary representations. Apart from the aforementioned difficulties related to the management of electronic data, further issues arise when addressing papery files, ordinarily filled in by both clerks and customers. Indeed, document filling is subjected to erroneous data-entry, misspelled terms, transposition over- sights, inconsistent abbreviations, lack of attribute values and so forth. These further increase the heterogeneity of bank data by making the individual records of a same textual document (e.g. a collection of personal demographic information) appear with a varying structure.

Impedance mismatch calls for effective preprocessing methodologies and tools, that enforce data quality and consistency within a centralized data warehouse, so that to ultimately ensure the reliability of business-intelligence results. In particular, a schema reconciliation technique is required to identify a common field structure for heterogeneous bank data, in order to exploit the mature relational technology for more effective information management. Also, a suitable methodology for handling with data duplication is useful to discover synonymies in the data, i.e. apparently different records that, as a matter of facts, refer to a same real-world entity. This would allow to perform tasks such as summing up the singularities (insolvencies, for instance) of

seemingly distinct individuals and associating them to a unique customer, so that to mark the latter as seriously critical.

In this section, we present a decision support system, that enables risk analysis in a banking environment. The core of the system is a centralized data warehouse, allowing the end user to perform predictive, multidimensional analysis, over millions of transactions from customers of a bank agency.

Innovative preprocessing methodologies, based on data mining techniques, are exploited to populate the warehouse. These are highly effective at dealing with schema reconciliation and duplicate detection issues and, hence, assure a reliable integration and consolidation of heterogeneous bank data into a unique archive.

The overall architecture of the decision support system is illustrated in fig. 7.14.

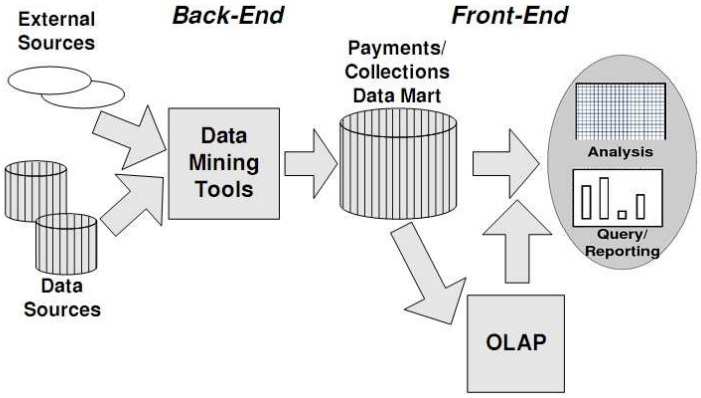


Fig. 7.14. Bank intelligence system anatomy

Focus, here, is on its main constituting components, namely the payments/collections (PC) data mart and the suite of data min- ing tools, that are discussed next. In our modeling, the focus is on business transactions: these corresponds to facts, and the chosen measures are the overall number and countervalue. Four the dimensions identified:

- *creditor*, i.e. individuals collecting transactions;
- *debtor*, i.e. people required to pay for transactions;
- *transaction category*, i.e. the class of financial service delivered to the customer;
- *state*, i.e. current transaction status (e.g. payed, unsolved, protested, under judicial action).

Information from each transaction comes from several sources, such as legacy operational databases and papery documents. The design of the PC data mart plays a key role in the process of decision making. In particular, the identification of its dimensions actually determines the capability of the overall system to answer meaningful business queries. However, several difficulties arise while constructing the foresaid warehouse. More specifically, these divide into syntactic and semantic issues. The former involve two major problems, namely schema reconciliation and data reconciliation.

In our bank scenario, the requirement for schema reconciliation mainly follows from the textual format of the personal information concerning both debtors and creditors. This information may not be uniformly formatted and, hence, its constituting records may apparently conform to different schemas. In practice, there is a similar situation to that showed in 7, where order of appearance of personal information attributes across the individual lines of text may not be fixed. Indeed, the collection of personal information may be fragmented over disparate data sources, which further exacerbates the aforementioned difficulties.

The requirement for data reconciliation originates from the fact that demographic information about creditors/debtors does not follow a canonical encoding, but differentiates on the basis of the category of the underlying transaction. Thus, certain transactions may not need address information or SSN codes, whereas others allow for such information to be manually inserted within the enterprise registry through papery forms. As a result, this heterogeneity yields lack of certainty in the identification of an individual.

As to the semantic issues behind the construction of our PC Data mart, the choice of how to model dimensions is crucial to ensure flexibility and efficiency at querying time. In our setting, modeling the *States* dimension (which groups possible transaction states, such as payed, unsolved, and so forth) is particularly critical. In particular, we identified two possible ways of modeling such states:

1. via a single dimension *state* containing all the admissible states for a transaction;
2. by adopting a single dimension for each possible *state*.

We chose the second option for two main reasons. First, since OLAP queries were defined by exploiting the Multidimensional Expressions (MDX, see [65]) technology, it simplifies the definition of MDX queries and avoids the well-known double count problem [78]. Notice that MDX is not strictly required in our applicative setting. Nevertheless, we still exploited MDX expressions, since they considerably simplify the definition of multidimensional queries.

Secondly, adopting a single dimension for each possible state also ensures a high efficiency since the aggregates are all computed at the same time, i.e. as soon as the data cube is built. However, in principle, such a choice may limit the incrementality of data warehouse. As a matter of fact, if a new state

occurs for a given transaction (e.g., it passes from being originally unsolved to the payed status), all the tuples already inserted into the data warehouse need be updated. Worst, if the bank organization decides to add a new state to the set of those already identified, the reorganization of the data warehouse may be required, which consequently implies acting on all data marts built on top of it. Nevertheless, state updates are rarely occurring in practical applications and this justifies our design choice.

In the remaining part of this section, we discuss an efficient clustering technique, that allows to discover groups of 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 indexing technique is based on a hashing scheme exploiting a family of Locality-Sensitive hashing functions [46].

Clustering approach

Let $\mathcal{M} = \{a_1, a_2, \ldots, a_m\}$ be an item domain. We assume m to be very large: typically, $\mathcal{M}$ represents the set of all possible strings available from a given alphabet. Moreover, we hypothesize 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. For the sake of simplicity, we view a tuple μ as a subset of $\mathcal{M}$. For instance, the tuple below represents a record of personal information on an individual.

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

Notice that, a more appropriate representation can take into account a relational schema in which each tuple fits. However, the results which follow can be easily generalized to such a similar context.

In this setting, our clustering approach to de-duplication can be roughly stated as the problem of 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. 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, $O(N)$.

A key intuition is that, in such a situation, it suffices to compare few *close* neighbors in order to obtain the appropriate cluster membership. Therefore, cluster membership can be detected by means of a minimal number of comparisons, by considering only some relevant neighbors for each new tuple, efficiently extracted from the current database through a proper retrieval method.

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 $\mathcal{DB}$.

The retrieval of neighbors in our clustering approach can be performed by resorting to an indexing scheme that supports the execution of similarity queries, 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 [23].

Hierarchical approximate hashing based on q-Grams

Our objective consists in defining a suitable hash-based index. In particular, we aim at defining an effective and efficient 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 this purpose, we combine two different techniques: (*i*) the adoption of hash functions based on the notion of *minwise independent permutation* [15], for bounding the probability of collisions, and (*ii*) the use of q-grams (i.e., contiguous substrings of size q) for a proper approximation of the similarity among string tokens [48].

Informally, 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. Clearly, such a function H can be exploited in order to assign similar tuples to a same bucket. The theory of minwise independent permutations [15] can employed to this purpose. Indeed, 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.

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 $\Pr[\min(\pi(X)) = \min(\pi(Y))] = (|X \cap Y|)/(|X \cup Y|)$. 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 for a suitable indexing approach. When $\mathcal{M}$ contains string tokens (as it usually happens in a typical information integration setting), the features of interest of a given token a can be represented by the q-grams of a. It has been shown [48] 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. 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 (resp. $a_2/a_3/a_4$) represents an "approximately common" term.
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 $feat(a_i)$ and $feat(a_j)$ represent two sets of features for tokens a_i and a_j respectively, then $d_{\mathcal{M}}(a_i, a_j) = 1 - |feat(a_i) \cap feat(a_j)|/|feat(a_i) \cup feat(a_j)|$. Both sets $feat(a_i)$ $feat(a_j)$ can be defined in terms of q-grams. 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. At 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. Instead, at the second level, the set $rep(\mu) = \{H^l(a)|a \in \mu\}$ 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.

A key point is the definition of a proper family of minwise independent permutations upon which to define the hash functions. In our approach we exploit the family of "practically" minwise independent permutations $\pi(x) = ((a \cdot c(x) + b) \bmod p)$ (introduced in [15]), 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).

Moreover, in order to further act on the randomness of the encoding, we exploit in disjunctive manner h different encodings $(H_1^l, \ldots, H_h^l)$ at first encoding level, and k different encodings $(H_1^u, \ldots, H_k^u)$ in conjunctive manner at second level of encoding. By varying h and k parameters, we act on the effectiveness (recall and precision respectively) of our hashing approach [23].

7.3.2 Evaluation and discussion

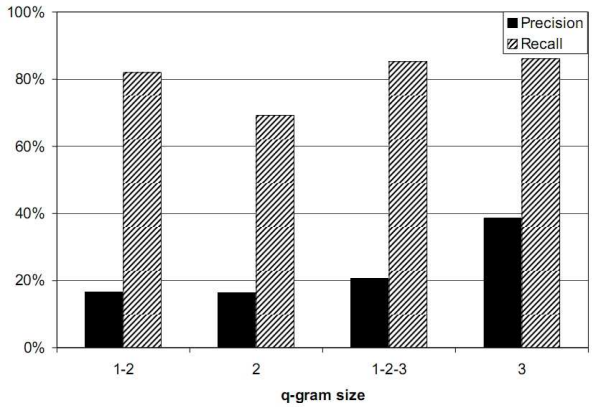
We here provide the results of an empirical evaluation performed to investigate the effectiveness of our methodologies for dealing with data duplication (schema reconciliation effectiveness was already tested in 7).

Our tests are performed by considering the `Addresses` data-set, the real-life demographic database, consisting of registry information about the issue-holders of credit situations in the afore described bank scenario. Such a data-set is of particular interest, since it contains several fragments of noisy data. We concentrate on a subset of 24,000 sequences of the `Addresses` data-set, where each sequence exhibits an average of 8 tokens per sequence. The schema to reconcile consists of the fields *Name*, *Address*, *Zip*, *State/Province*, and *City*.

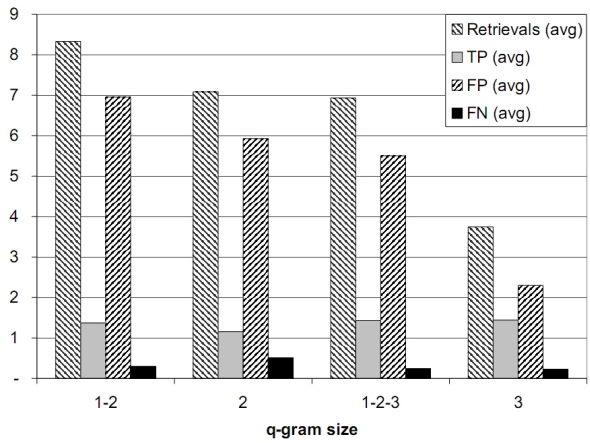
The discussion on the hash-based approach for the de-duplication shows that the effectiveness of the approach relies on proper values of h (number of minwise functions at first level of encoding) and k (number of minwise functions at second level of encoding). Suitable values of such parameters fixing a high correspondence between the retrieved and the expected neighbors of a tuple were investigated [23], where we justified and showed that best performances are obtainable for $h = k = 5$. Again, experiments are conducted over `Addresses` data-set.

For a generic tuple μ we newly consider the number TP (i.e., the tuples which are retrieved and that belong to the same cluster of μ), and compare it to the number of FP (i.e., tuples retrieved without being neighbors of μ), and FN (i.e., neighbors of μ which are not retrieved). As global indicators we exploit the average *Precision* and *Recall* per tuple, i.e. $P = \frac{1}{N} \sum_{\mu \in \mathcal{DB}} \frac{TP}{TP+FP}$ and $R = \frac{1}{N} \sum_{\mu \in \mathcal{DB}} \frac{TP}{TP+FN}$, where N denotes the number of tuples in $\mathcal{DB}$. The values of such quality indicators influence the effectiveness of the our clustering scheme. In general, high values of precision allow for correct de-duplication. When precision is low, the clustering method can be effective only if recall is high.

More precisely, if we fix $h = k = 5$, fig. 7.15(a) shows the results obtained for precision and recall by using different values of q-gram size q, whereas fig. 7.15(b) summarizes the average number of retrievals and quality indices. Notice that, in fig. 7.15(a), $q = 3$ allows a recall quite high even if precision is low (thus allowing for a still effective clustering). Moreover, fig. 7.15(b) indicates that, by using the same parameter combination, we are able to keep the average number of retrievals low, thus guaranteeing a good scalability of the approach.



(a) Precision and recall



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

Fig. 7.15. Results on real data using different q-gram sizes

In conclusion, this case study showed how business-targeted data warehouses require complex efforts for their definition and setting. In particular, enterprise data require to be suitably cleaned, reconciled and integrated, being originally stored in a federation of operational databases, scattered across various departments of an organization (e.g. the accounts, production and R&D departments). Preprocessed data are stored into data warehousing infrastructures, that provide fast access facilities to such data as well several functionalities for efficient query processing and data analysis.

Starting from a real-life scenario involving risk analysis in banking organizations, we showed how effective data mining techniques can effectively ease the process of identifying and solving critical factors which affect the quality

of the data warehouse itself. In particular, we devised two effective techniques for schema and data reconciliation, and applied them to the aforementioned scenario. As a result, this enables and expedites a number of management and/or analytical tasks, such as well informed decision making, learning models of customer behavior and future trend prediction.

Clearly, the application scenario described here is not the only one where the proposed techniques apply. To this purpose, we plan to test their effectiveness in other domains, where failure in either schema or data reconciliation may significantly affect the quality of analysis results.

Part IV

Conclusion

8

Conclusion

8.1 Defining and realizing a knowledge discovery framework

The definition of a unifying framework, wherein to accommodate and combine mining and data-processing tasks as components of a multi-step analytical process, has not received the necessary attention: despite some preliminary ideas, it is still an open issue without a predominant proposal. The absence of such a framework is a primary limitation for the real-world applications of data analysis, where it rarely happens that a single pattern-mining activity suffices to meet the underlying analytical requirements: the process of knowledge discovery can actually never be reduced to a single pattern-mining activity.

A main contribution of this thesis was the definition and implementation of a knowledge discovery framework built upon formal basis. In effect, the operational semantics of the framework was founded on the 2W Model, a theoretic model that views the knowledge discovery process as a progressive combination of mining and querying operators. The aforementioned theoretic model, was introduced after having recognized some common high-level problems that arise when attempting to define a KD framework, and consequently having identified some basic requirements such a framework must have. The essence of the defined 2W Model framework is the interaction between *data world* and *model world*, thus the proposed framework supports an operations flow in which mining tasks can be performed, and the resulting models can be further exploited to leverage the overall discovery process.

A possible implementation of the 2W Model framework was also proposed. In defining such implementation, easy of use and fully extensibility was two main requirements, but we also solved real-world problems such as managing complex data types and efficiently storing and accessing huge amounts of data. Besides being completely compliant with the theoretic 2W Model, the proposed implementation also allows the analyst to easily examine and in-

spect both data and built models, using statistics, charts, and various models visualizations metaphors.

There are some challenging issues, that are worth further research. Foremost, the identification of a compact 2W Model algebra, consisting of a fixed, minimal set of operators. Analogously to the case of the 3W Model framework, this is useful in two respects: (i) the possibility of expressing the required patterns via suitable combinations of such basic operators, rather than relying on an arbitrary number of task-oriented mining operators; (ii) the development of a solid theoretical background concerning expressiveness and complexity results. As far as optimization perspectives are concerned, the development of strategies for decoupling specification from execution and optimizing processing plans would increase the overall performance of the proposed framework engine. Finally, it is worth noticing that the 2W Model can in principle be adopted as a foundation for the definition of the procedural semantics of a data mining query language.

8.2 The RecBoost application

Another contribution of this thesis was 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. Also, a comparative analysis with state-of-the-art alternative approaches reveals the following two main arguments in favor of RecBoost:

- due to the variable number of classification stages, RecBoost gives the user better control over the trade-off between accuracy (i.e. the proportion of correctly classified tokens w.r.t. the classification behavior of the overall RecBoost system) and recall (i.e. the proportion of correctly classified tokens w.r.t. the actual tokens to reconcile). In practice, the user can choose a classifier with a trade-off satisfying the requirements of the specific application;
- the generic RecBoost classifier is easier to interpret than existing methods such as DATAMOLD [12] and Mallet [69], since it produces symbolic rules using vocabulary from a domain-specific ontology.

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, we plan to 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. Finally, we intend to examine the exploitation of RecBoost in the context of the Entity Resolution process, to the purpose of properly filling in missing fields and rectifying both erroneous data-entry and transpositions oversights.

References

1. A. Abdulghani, T. Imielinski, and A. Virmani. Disovery board application programming interface and query language for database mining. In *KDD96 (Portland, Ore.)*, pages 20–26, 1996.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. B. Adelberg. A tool for semi-automatically extracting semistructured data from text documents. In *ACM SIGMOD Conference on Management of Data*, 1998.
4. E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *ACM SIGKDD Conference On Knowledge Discovery and Data Mining*, pages 20–29, 2004.
5. R. Agrawal and R. Srikant. Mining generalized association rules. In *21th International Conference on on Very Large Databases*, pages 407–419, 1995.
6. T. Anand and R. Brachman. The process of knowledge discovery in databases: A human centered approach. In *AKDDM*, pages 37–58, 1996.
7. JSR-73 Expert Group. Java Data Mining API. Website. http:// www.jcp.org/ en/ jsr/ detail?id=73.
8. M. Baglioni and F. Turini. Mql: An algebraic query language for knowledge discovery. In *8th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence (AI∗IA 2003)*, pages 225–236, 2003.
9. R.A. Baxter, L. Gu, D. Vickers, and et al. Record linkage: Current practice and future directions. In *Technical report, CSIRO Mathematical and Information Sciences*, 2003.
10. M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational expressive power of constraint query languages. In *Journal of the ACM*, pages 1–34, 1998.
11. Bernhardt, S. Chaudhuri, U.M. Fayyad, and A. Netz. Integrating data mining with sql databases: Ole db for data mining. In *IEEE ICDE Int. Conf. on Data Engineering*, pages 379–387, 2001.
12. V.R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *ACM SIGMOD Conference on Management of Data*, pages 175–186, 2001.
13. J.F. Boulicaut, M. Klemettinen, and H. Mannila. Modeling kdd processes within the inductive database framework. In *DaWaK International Conference on Data Warehousing and Knowledge Discovery*, pages 293–302, 1999.

14. E. Brill. Transformation-based error-driven learning and natural language processing: A cased study in POS tagging. In *Computational Linguistics*, pages 543–565, 1995.
15. A.Z. Broder, M. Charikar, A.M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *30th Symp. on the Theory of Computing*, pages 327–336, 1998.
16. W. Buntine. Graphical models for discovering knowledge. In *AKDDM*, pages 59–82, 1996.
17. CRISP-DM: Step by-step data mining guide. Website. http:// www.crisp-dm.org.
18. T. Calders, L.V.S. Lakshmanan, R.T. Ng, and J. Paredaens. Expressive power of an algebra for data mining. In *ACM Transactions on Database Systems*, pages 1169–1214, 2006.
19. M.E. Califf and R.J. Mooney. Relational learning of pattern-match rules for information extraction. In *15th National Conference on Artificial Intelligence*, pages 328–334, 1998.
20. B. Carey and C. Marjaniemi. A methodology for evaluating and selecting data mining software. In *Thirty-second Annual Hawaii International Conference on System Sciences*, 1999.
21. E. Cesario, F. Folino, A. Locane, G. Manco, and R. Ortale. Recboost: A supervised approach to text segmentation. In *SEBD - Italian Symposium on Advanced Database Systems*, 2005.
22. E. Cesario, F. Folino, A. Locane, G. Manco, and R. Ortale. Boosting text segmentation via progressive classification. In *KAIS - Knowledge and Information Systems*, 2008.
23. E. Cesario, F. Folino, G. Manco, and L. Pontieri. An incremental clustering scheme for duplicate detection in large databases. In *9th Int. Symp. on Database Engineering and Applications*, pages 89–95, 2005.
24. S. Chauduri. Data mining and database systems: where is the intersection? In *Bulletin of the Technical Committe e on Data Engineering*, pages 4–8, 1998.
25. CLEMENTINE. Website. http:// www.spss.com/ clementine/.
26. M. Cochinwala, S. Dalal, and A.K. et al Elmagarmid. Record matching: Past, present and future. In *ACM Computing Surveys*, 2003.
27. W.W. Cohen. Learning to classify english text with ilp methods. In *Advances in Inductive Logic Programming*, pages 124–143, 1996.
28. G. Costa, F. Folino, A. Locane, G. Manco, and R. Ortale. Data mining for effective risk analysis in a bank intelligence scenario. In *ICDE Workshop on Data Mining and Business Intelligence*, 2007.
29. M. Craven, S. Ray, and M. Skounakis. Hierarchical hidden markov models for information extraction. In *18th International Joint Conference on Artificial Intelligence*, pages 427–433, 2003.
30. L. De Raedt. A logical database mining query language. In *ILP International Conference on Inductive Logic Programming*, pages 78–92, 2000.
31. J.S. Deogun, V.V. Raghavan, A. Sarkar, and H. Sever. Data mining: Research trends, challenges, and applications. In *Roughs Sets and Data Mining: Analysis of Imprecise Data*, 1997.
32. S. Dzeroski. Inductive logic programming for knowledge discovery in databases. In *AKDDM*, 1996.
33. S. Dzeroski. Multi-relational data mining: an introduction. In *SIGKDD Exploration Newsletter*, pages 1–16, 2003.

34. M.G. Elfeky, S.A. Fouad, and A.A. Saad. Odmql: Object data mining query language. In *Objects and Databases*, pages 128–140, 2000.
35. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The kdd process for extracting useful knowledge from volumes of data. In *Communications of the ACM*, pages 27–34, 1996.
36. U. Fayyad, G. Piatetsky-shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *2nd Int. Conf. on Knowledge Discovery and Data Mining*, 1996.
37. U.M. Fayyad, G. PiatetskyShapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In *AKDDM*, pages 1–30, 1996.
38. F. Flesca, G. Manco, and E. Masciari. Web wrapper induction: A brief survey. In *AI Communications*, pages 57–61, 2004.
39. Magic Quadrant for Customer Data-Mining Applications. Herschel, g. In *Gartner RAS Core Research Note*, 2007.
40. E. Frank and I.H. Witten. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan and Kaufmann, 2000.
41. D. Freitag. Toward general-purpose learning for information extraction. In *17th National Conference on Computational Linguistics*, pages 404–408, 1998.
42. D. Freitag, A. McCallum, and F. Pereira. Maximum entropy markov models for information extraction and segmentation. In *17th International Conference on Machine Learning*, pages 591–598, 2000.
43. Y. Fu, J. Han, K. Koperski, and W. et al Wang. Dbminer: A system for mining knowledge in large relational databases. In *International Conference on Data Mining and Knowledge Discovery*, pages 250–255, 1996.
44. Y. Fu, J. Han, K. Koperski, and W. et al Wang. Dmql: A data mining query language for relational databases. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1996.
45. I. Geist and K. Sattler. Towards data mining operators in database systems: Algebra and implementation. In *DBFusion International Workshop on Databases, Documents, and Information Fusion*, 2002.
46. A. Gionis, P. Indyk, and Motwani R. Similarity search in high dimensions via hashing. In *25th Int. Conf. on Very Large Data Bases*, pages 518–529, 1999.
47. M. Goebel and L. Gruenwakd. A survey of data mining and knowledge discovery software tools. In *SIGKDD Explorations*, pages 20–33, 1999.
48. L. et al Gravano. Approximate string joins in a database (almost) for free. In *27th Int. Conf. on Very Large Data Bases*, pages 518–529, 2001.
49. J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, San Mateo, CA, 2000.
50. D. Hand, H. Mannila, and Smyth. *Principles of Data Mining.* MIT Press, Cambridge, MA, 2001.
51. M.A. Hernández and J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. In *Data Mining and Knowledge Discovery*, pages 9–37, 1998.
52. H. Hirsh and T. Imielinski. Query-based approach to database mining. In *Technical report, Rutgers University*, 1993.
53. W. Hsu, B. Liu, and Y. Ma. Integrating classification and association rule mining. In *4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 80–86, 1998.
54. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. In *Communications ACM*, pages 58–64, 1996.

55. T. Imielinski and A. Virmani. Msql: A query language for database mining. In *Data Mining and Knowledge Discovery*, pages 373–408, 1999.
56. Infor. Website. http:// go.infor.com/ inforcrm/.
57. Fair Isaac. Website. http:// www.fairisaac.com/ fic/ en.
58. H. Jagadish, L. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses? In *VLDB*, pages 530–541, 1999.
59. T. Johnson, L. Lakshmanan, and R.T. Ng. The 3w model and algebra for unified data mining. In *International Conference on Very Large Data Bases (VLDB 2000)*, pages 21–32, 2000.
60. M. Junker, M. Rinck, and M. Sintek. Learning for text categorization and information extraction with ilp. In *Learning Language in Logic*, pages 247–258, 2001.
61. V. Kumar, P.N. Tan, and M. Steinbach. *Introduction to Data Mining*. Addison-Wesley, 2006.
62. J. Kupiec. Robust part-of-speech tagging using a hidden markov model. In *Computer Speech and Language*, pages 225–242, 1992.
63. KXEN. Website. http:// www.kxen.com.
64. J.D. Lafferty, A. McCallum, and F.C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *18th International Conference on Machine Learning*, pages 282–289, 2001.
65. MDX: Microsofts language for OLAP. Website. http:// www.microsoft.com.
66. A. Locane, G. Manco, R. Ortale, and E. Ritacco. A 2w algebra model for knowledge discovery. In *Technical report ICAR-CNR*, 2008.
67. C.D. Manning and C. Schultze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
68. L. Marquez, L. Padro, and H. Rodriguez. A machine learning approach to pos tagging. In *Machine Learning*, pages 59–91, 2000.
69. A. McCallum. Mallett: Advanced machine learning for language toolkit. In *http://mallet.cs.umass.edu*, 2002.
70. IBM DB2 Intelligent Miner. Website. http:// www-01.ibm.com/ software/ data/ iminer/.
71. Oracle Data Mining. Website. http:// www.oracle.com/ technology/ products/ bi/ odm.
72. SQL Server Data Mining. Website. http:// www.sqlserverdatamining.com/ ssdm/.
73. B. Mitbander, W.M. Shen, K. Ong, and C. Zaniolo. Metaqueries for data mining. In *Advances in Knowledge Discovery and Data Mining*, pages 375–398, 1996.
74. T. Mitchell. *Machine Learning*. Mc Graw-Hill, 1997.
75. T. Morzy and M. Zakrzewicz. Sql-like language for database mining. In *ADBIS First East-European Symposium on Advances in Databases and Information Systems*, pages 311–317, 1997.
76. S. Mukherjee and I.V. Ramakrishnan. Taming the unstructured: Creating structured content from partially labeled schematic text sequences. In *12th CoopIS/DOA/ODBASE International Conference*, pages 909–926, 2004.
77. KDD Nuggets. Website. http:// www.kdnuggets.com.
78. T.B. Pedersen, C.S. Jensen, and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. In *Information Systems*, page 383423, 2001.
79. G. Piatetsky-Shapiro. Knowledge discovery in real databases. In *AI Magazine*, pages 68–70, 1991.

80. The DataMining Group. PredictiveModelMarkup Language (PMML). Website. http:// http://www.dmg.org.
81. D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, San Francisco, 1999.
82. A. Romei, S. Ruggieri, and F. Turini. Kddml: a middleware language and system for knowledge discovery in databases. In *Data Knowledge and Engineering*, pages 197–220, 2006.
83. SAS. Website. http:// www.sas.com.
84. C. Seidman. Data mining with microsoft sql server 2000. In *Technical Reference. Microsoft Press*, 2001.
85. S. Soderland. Learning information extraction rules for semi/structured and free text. In *Machine Learning*, pages 233–272, 1999.
86. Angoss Software. Website. http:// www.angoss.com.
87. Portrait Software. Website. http:// www.portraitsoftware.com.
88. OLE DB DM Specifications. Website. http:// www.microsoft.com/ data/ oledb/ dm/.
89. SPSS. Website. http:// www.spss.com.
90. Exeura technical-scientific reports on Data Mining Suite and Rialto. Techinical reports. http:// www.exeura.com.
91. ThinkAnalytics'. Website. http:// www.thinkanalytics.com.
92. Unica. Website. http:// www.unica.com.
93. H. Wang and C. Zaniolo. Atlas: A native extension of sql for data mining. In *Third SIAM International Conference on Data Mining*, 2003.
94. WEKA. Website. http:// www.cs.waikato.ac.nz/ ml/ weka/.
95. W.E. Winkler. The state of record linkage and current research problems. In *Technical report, Statistical Research Division, U.S. Census Bureau*, 1999.