Learning classifiers from distributed, semantically heterogeneous, autonomous data sources

by

Doina Caragea

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Vasant Honavar, Major Professor
Dianne Cook
Drena Dobbs
David Fernandez-Baca
Leslie Miller

Iowa State University

Ames, Iowa

2004

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Doina Caragea

has met the dissertation requirements of Iowa State University

Major Professor

For the Major Program

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I express my gratitude to my advisor Dr. Vasant Honavar for guiding the research presented in this dissertation throughout my Ph.D. student years. He has been a constant source of motivation and encouragement. He helped me to develop my own views and opinions about the research I undertook. I thank him for being always accessible and for providing invaluable feedback on my work. I also thank him for providing funding for my research and for helping me to receive an IBM Fellowship two years in a row. Thanks for organizing the AI seminar which brought up thoughtful discussions and helped me to broaden my views about Artificial Intelligence. Most importantly, I thank him for his friendship and for his encouragement when I felt confused or overwhelmed.

I give my warm thanks to Dr. Dianne Cook for introducing me to the wonderful world of visualization, for being a close collaborator and a model for me, as well as well as a good friend. Thanks also go to Dr. Drena Dobbs for introducing me to the world of molecular biology and motivating me to pursue a minor in bioinformatics, and to Dr. Leslie Miller and Dr. David Fernandez-Baca for being the first two people that I interacted with in my first semester at Iowa State University. They both helped me overcome my fears of graduate school. I thank everybody in my committee for fruitful interactions and for their support.

I am grateful to Adrian Silvescu for motivating me to go to graduate school, for collaborating with me on several projects, for his great ideas and enthusiasm, and for being one of my best friends ever.

Thanks to all the students who were present in the Artificial Intelligence lab at Iowa State University while I was there. They were great colleagues and provided me with a friendly environment to work in. Thanks to Jyotishman Pathak for closely collaborating with me on ontology-extended data sources and ontology-extended workflow components during the last

year of my Ph.D. studies. Thanks to Facundo Bromberg for interesting discussions about multi-agent systems. Thanks to Dae-Ki Kang for letting me use his tools for generating taxonomies and for teaching me about AT&T graphviz. Thanks to Jun Zhang for the useful discussions about partially specified data. Thanks to Jie Bao for discussions and insights about ontology management. Thanks to Changui Yan and Carson Andorf for useful discussions about biological data sources. Thanks to Oksana Yakhnenko for helping with the implementation of AirlDM. Finally, thanks to Jaime Reinoso-Castillo for the first INDUS prototype.

It has been an honor to be part of the Computer Science Department at Iowa State University. There are many individuals in Computer Science Department that I would like to thank for their direct or indirect contribution to my research or education. Special thanks to Dr. Jack Lutz for introducing me to the fascinating Kolmogorov Complexity. I thoroughly enjoyed the two courses and the seminars I took with Dr. Lutz.

I also thank the current as well as previous staff members in Computer Science Department, especially Linda Dutton, Pei-Lin Shi and Melanie Eckhart. Their kind and generous assistance with various tasks was of great importance during my years at Iowa State University.

I am grateful to Dr. John Mayfield for financial support from the Graduate College and to Sam Ellis from IBM Rochester for his assistance in obtaining the IBM graduate fellowship.

The research described in this thesis was supported in part by grants from the National Science Foundation (NSF 0219699) and the National Institute of Health (NIH GM066387) to Vasant Honavar.

Above all, I am fortunate to have family and friends that have provided so much support, encouragements and love during my Ph.D. years and otherwise. Thanks to my parents, Alexandra and Paul Caragea, to my sister Cornelia Caragea and to my cousin Petruta Caragea. Thanks to my friends Pia Sindile, Nicoleta Roman, Simona Verga, Veronica Nicolae, Calin Anton, Liviu Badea, Mircea Neagu, Marius Vilcu, Cristina and Marcel Popescu, Petrica and Mirela Vlad, Anna Atramentova, Laura Hamilton, Carol Hand, Barbara Gwiasda, Emiko Furukawa, Shireen Choobineh, JoAnn Kovar, Mike Collyer, and especially to Charles Archer for helping me believe that I was able to complete this thesis.

ABSTRACT

Recent advances in computing, communications, and digital storage technologies, together with development of high throughput data acquisition technologies have made it possible to gather and store large volumes of data in digital form. These developments have resulted in unprecedented opportunities for large-scale data-driven knowledge acquisition with the potential for fundamental gains in scientific understanding (e.g., characterization of macromolecular structure-function relationships in biology) in many data-rich domains. In such applications, the data sources of interest are typically physically distributed, semantically heterogeneous and autonomously owned and operated, which makes it impossible to use traditional machine learning algorithms for knowledge acquisition.

However, we observe that most of the learning algorithms use only certain statistics computed from data in the process of generating the hypothesis that they output and we use this observation to design a general strategy for transforming traditional algorithms for learning from data into algorithms for learning from distributed data. The resulting algorithms are provably exact in that the classifiers produced by them are identical to those obtained by the corresponding algorithms in the centralized setting (i.e., when all of the data is available in a central location) and they compare favorably to their centralized counterparts in terms of time and communication complexity.

To deal with the semantical heterogeneity problem, we introduce ontology-extended data sources and define a user perspective consisting of an ontology and a set of interoperation constraints between data source ontologies and the user ontology. We show how these constraints can be used to define mappings and conversion functions needed to answer statistical queries from semantically heterogeneous data viewed from a certain user perspective. That is further used to extend our approach for learning from distributed data into a theoretically

sound approach to learning from semantically heterogeneous data.

The work described above contributed to the design and implementation of AirlDM, a collection of data source independent machine learning algorithms through the means of sufficient statistics and data source wrappers, and to the design of INDUS, a federated, query-centric system for knowledge acquisition from distributed, semantically heterogeneous, autonomous data sources.

1 INTRODUCTION

1.1 Motivation

Recent advances in computing, communications, and digital storage technologies, together with development of high throughput data acquisition technologies have made it possible to gather and store large volumes of data in digital form. For example, advances in high throughput sequencing and other data acquisition technologies have resulted in gigabytes of DNA, protein sequence data, and gene expression data being gathered at steadily increasing rates in biological sciences; organizations have begun to capture and store a variety of data about various aspects of their operations (e.g., products, customers, and transactions); complex distributed systems (e.g., computer systems, communication networks, power systems) are equipped with sensors and measurement devices that gather and store a variety of data for use in monitoring, controlling, and improving the operation of such systems.

These developments have resulted in unprecedented opportunities for large-scale data-driven knowledge acquisition with the potential for fundamental gains in scientific understanding (e.g., characterization of macro-molecular structure-function relationships in biology) in many data-rich domains. To exploit these opportunities scientists at different institutions need to collaborate and share information and findings in a field or across various research fields [Hendler, 2003]. Thus, researchers working at one level of a problem may benefit from data or results developed for a different level of that problem or even for a different problem.

However, more often than not, it is not easy for a scientist to be able to use the information obtained from a different scientific community. Furthermore, even scientists working on the same problem at different institutions find it difficult to combine their results. These difficulties arise because of the large volume of information that would need to be moved around or

because of the constraints imposed by the autonomy of the data collected by a particular institution (e.g., privacy constraints). Even in cases when data can be shared, the heterogeneity of the data collected by different scientific communities or organizations brings several difficulties. This heterogeneity could be in terms of structure (relational databases, flat files, etc.) or content (different ontological commitments, which means different assumptions concerning the objects that exist in the world, the properties or attributes of the objects, the possible values of attributes, and their intended meaning) [Levy, 2000]. Thus, the current technology is not sufficient for the need of collaborative and interdisciplinary "e-Science" [e-Science, 2001], but fortunately, new technologies are emerging with the potential to revolutionize the ability of scientists to do collaborative work [Hendler, 2003].

Among these, a new generation of Web technology, the Semantic Web [Berners-Lee *et al.*, 2001], aims to support seamless and flexible access and use of semantically heterogeneous, networked data, knowledge, and services. Thus, the Semantic Web is supposed to improve communication between people using differing terminologies, to extend the interoperability of databases, and to provide new mechanisms for the support of agent-based computing in which people and machines work more interactively, making possible a new level of interaction among scientific communities [Hendler, 2003].

Examples of scientific domains that have started to use the Semantic Web include biological [AMIAS, 2002], environmental [SWS, 2002] and astronomical [Szalay, 2001] domains, which are trying to link together various heterogeneous resources. Even mathematical sciences [MONET, 2004] are exploring the use of the Semantic Web for making mathematical algorithms Web-accessible from a variety of software packages.

The e-Science initiative in the UK [e-Science, 2001] brings together research scientists and information technologists in an effort to make possible the Semantic Web vision in science, and recently resulted in an initiative to unite the Semantic Web and Grid computing [Euroweb, 2002] as a step towards achieving the goals of the collaborative e-Science.

It is worth noting that the Semantic Web vision cannot be achieved without exploiting artificial-intelligence technologies in addition to the Semantic Web [Berners-Lee *et al.*, 2001]. Hence, there has been significant interest in Semantic Web "agents" that can answer queries

based on information from Web pages and heterogeneous databases and pass them to programs for analysis [Hendler, 2003].

Against this background, this dissertation explores the problem of automated or semi-automated data driven knowledge acquisition (discovery of features, correlations, and other complex relationships and hypotheses that describe potentially interesting regularities from large data sets) from distributed semantically heterogeneous autonomous data sources (see Figure 1.1).

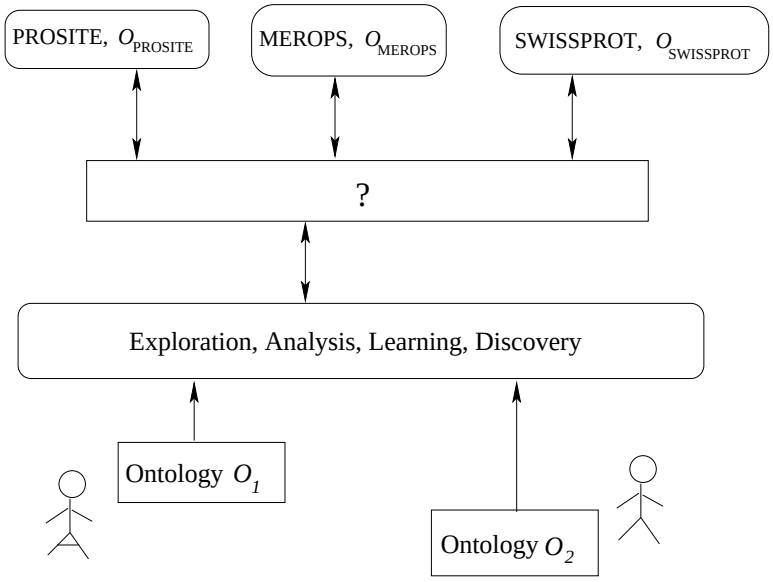

Figure 1.1 Example of a scenario that calls for knowledge acquisition from autonomous, distributed, semantically heterogeneous data source - discovery of protein sequence-structure-function relationships using information from PROSITE, MEROPS, SWISSPROT repositories of protein sequence, structure, and function data. O_1 and O_2 are two user ontologies

The major contributions of this dissertation include:

- **A general strategy for design of algorithms for learning classifiers from distributed data** [Caragea *et al.*, 2004d]
- **A general framework for design of algorithms for learning classifiers from semantically heterogeneous data** [Caragea *et al.*, 2004b]
- **Design of a query answering engine** [Caragea *et al.*, 2004a]

- **An open source package containing data source independent machine learning algorithms** [Silvescu *et al.*, 2004b]

1.2 Traditional Machine Learning Limitations

Machine learning algorithms [Mitchell, 1997; Duda *et al.*, 2000] offer some of the most cost-effective approaches to automated or semi-automated knowledge acquisition in scientific domains. However, the applicability of current approaches to machine learning in emerging data rich applications in practice is severely limited by a number of factors:

- **Distributed Data Sources:** As mentioned above, data repositories are large in size, dynamic, and physically distributed. Consequently, it is neither desirable nor feasible to gather all of the data in a centralized location for analysis. Hence, there is a need for knowledge acquisition systems that can perform the necessary analysis of data at the locations where the data and the computational resources are available and transmit the results of analysis (knowledge acquired from the data) to the locations where they are needed [Honavar *et al.*, 1998]. In other domains, the ability of autonomous organizations to share raw data may be limited due to a variety of reasons (e.g., privacy considerations) [Agrawal and Srikant, 2000]. In such cases, there is a need for knowledge acquisition algorithms that can learn from statistical summaries of data (e.g., counts of instances that match certain criteria) that are made available as needed from the distributed data sources in the absence of access to raw data.

- **Heterogeneous Data Sources:** According to the Semantic Web [Berners-Lee *et al.*, 2001], the ontological commitments associated with a data source are determined by the intended use of the data repository (at design time). Furthermore, data sources that are created for use in one context often find use in other contexts or applications. Semantic differences among autonomously designed, owned, and operated data repositories are simply unavoidable. Effective use of multiple sources of data in a given context requires reconciliation of such semantic differences from the user's point of view. Because users often need to analyze data in different contexts from different perspectives, there is no

single privileged ontology that can serve all users, or for that matter, even a single user, in every context. Hence, there is a need for methods that can dynamically and efficiently extract and integrate information needed for learning (e.g., statistics) from distributed, semantically heterogeneous data based on user-specified ontologies and mappings between ontologies.

- **Autonomous Data Sources:** Data sources of interest are autonomously owned and operated. Consequently, they differ in their structure and organization (relational databases, flat files, etc.) and the operations that can be performed on the data source (e.g., types of queries: relational queries, restricted subsets of relational queries, statistical queries, keyword matches; execution of user-supplied code to compute answers to queries that are not directly supported by the data source; storing results of computation at the data source for later use) and the precise mode of allowed interactions can be quite diverse. Hence, there is a need for theoretically well-founded strategies for efficiently obtaining the information needed for learning within the operational constraints imposed by the data sources.

1.3 Our Approach

Our approach to the problem described above comes from revisiting the traditional formulation of the problem of learning from data and observing that most of the learning algorithms use only certain *statistics* computed from the data in the process of generating the hypotheses that they output [Kearns, 1998]. This yields a natural decomposition of a learning algorithm into two components: an *information extraction* component that formulates and sends a statistical query to a data source and a *hypothesis generation* component that uses the resulting statistic to modify a partially constructed hypothesis (and further invokes the information extraction component as needed) (see Figure 1.2).

In the light of this observation, an algorithm for learning from distributed data can be also decomposed into two components: (1) *information extraction from distributed data* and (2) *hypothesis generation*.

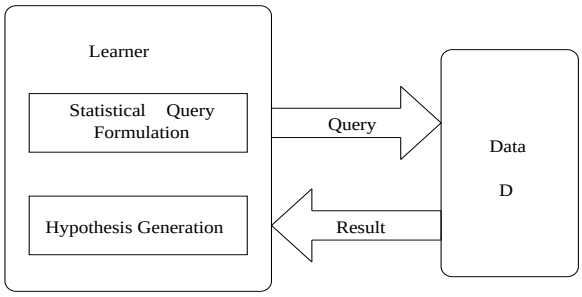

Figure 1.2 Learning revisited: identify sufficient statistics, gather the sufficient statistics and generate the current algorithm output

The information extraction from distributed data entails decomposing each statistical query q posed by the information extraction component of the learner into sub-queries $q_1, \cdots, q_K$ that can be answered by the individual data sources $D_1, \cdots, D_K$, respectively, and a procedure for combining the answers to the sub-queries into an answer for the original query q (see Figure 1.3). This yields a general strategy for transforming algorithms for learning from centralized data into exact algorithms for learning from distributed data (an algorithm L_d for learning from distributed data sets $D_1, \cdots, D_K$ is *exact* relative to its centralized counterpart L if the hypothesis produced by L_d is identical to that obtained by L from the complete data set D obtained by appropriately combining the data sets $D_1, \cdots, D_K$).

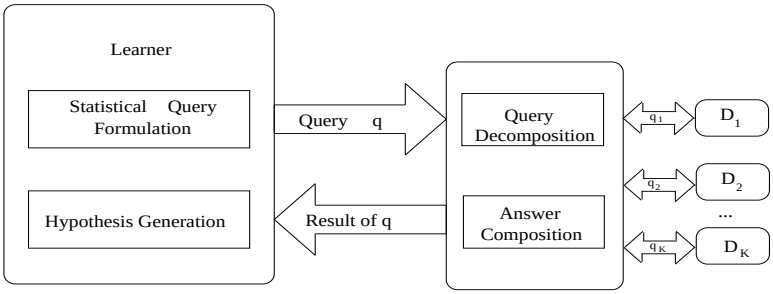

Figure 1.3 Exact learning from distributed data: distribute the statistical query among the distributed data sets and compose their answers

We consider two types of data fragmentation: *horizontal fragmentation* wherein (possibly overlapping) subsets of data tuples are stored at different sites and *vertical fragmentation* wherein (possibly overlapping) sub-tuples of data tuples are stored at different sites and we apply this strategy to design exact algorithms for learning Naive Bayes, Decision Trees, Threshold Functions, Support Vector Machines and k-NN classifiers from distributed data.

We compare the resulting algorithms with the traditional algorithms in terms of time and communication complexity.

In order to extend our approach to learning from distributed data (which assumes a common ontology that is shared by all of the data sources) into effective algorithms for learning classifiers from semantically heterogeneous distributed data sources, we develop techniques for answering the statistical queries posed by the learner in terms of the learner's ontology O from the heterogeneous data sources (where each data source D_k has an associated ontology O_k) (see Figure 1.4). Thus, we solve a variant of the problem of integrated access to distributed data repositories, the data integration problem [Levy, 2000], in order to be able to use machine learning approaches to acquire knowledge from semantically heterogeneous data.

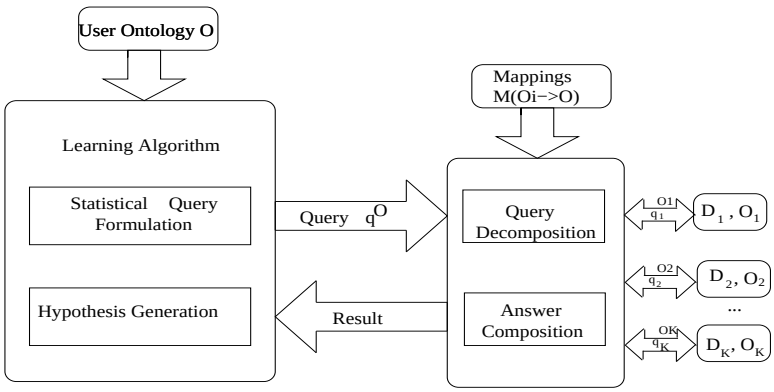

Figure 1.4 Learning from semantically heterogeneous distributed data: each data source has an associated ontology and the user provides a global ontology and mappings from the local ontologies to the global ontology

It can be seen that learning from distributed heterogeneous autonomous data sources reduces to the problem of developing sound and complete techniques for answering statistical queries from semantically heterogeneous data sources under a variety of constraints and assumptions motivated by application scenarios encountered in practice.

We define a statistical query language based on operators that are needed to formulate and manipulate statistical queries, and we design a query answering engine, that has access to a resource repository where all the information available in the system is registered. The engine uses these resources to decompose a query q into sub queries $q_1, \cdots, q_K$ that can be answered by the individual data sources $D_1, \cdots, D_K$ respectively, finding an optimal plan for

executing each of the sub queries q_i, and also a procedure for combining the answers to the sub queries into an answer for the original query q.

This builds on recent work on INDUS (see Figure 1.5), an ontology based federated, query-centric approach to information integration and learning from distributed, heterogeneous data sources. INDUS offers the functionality necessary to flexibly integrate information from multiple heterogeneous data sources and structure the results according to a user-supplied ontology. Learning algorithms are linked to the information integration component in INDUS, and thus users can perform learning from distributed heterogeneous data sources in a transparent way.

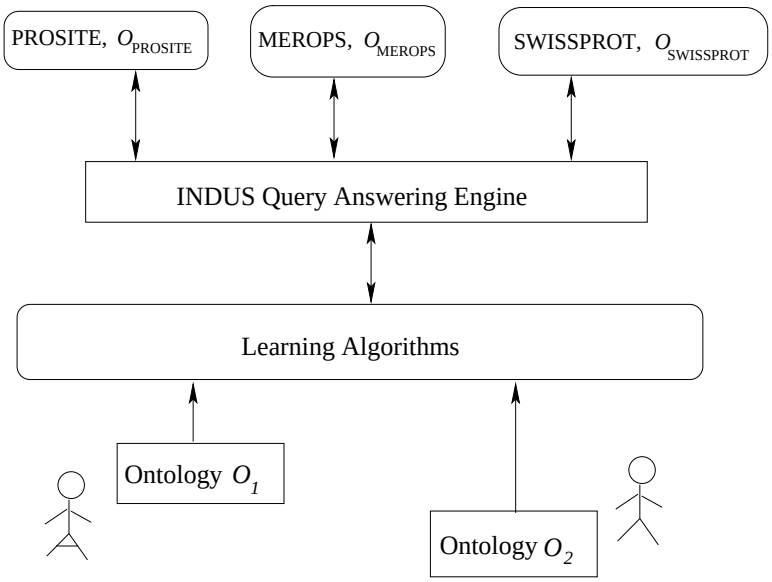

Figure 1.5 INDUS: INtelligent Data Understanding System

1.4 Literature Review

The work related to the research in this dissertation belongs to one of the following three categories: *distributed learning* [Liu *et al.*, 2004], *information integration* [Levy, 2000], or the combination of distributed learning and information integration, which we call *learning from semantically heterogeneous data* [Caragea *et al.*, 2004d].

1.4.1 Distributed Learning

Distributed learning (a.k.a., distributed data mining) has received considerable attention in literature [Liu *et al.*, 2004] in recent years. Work in this area can be reviewed from three points of view: *distributed learning algorithms*, *architectures and systems for distributed learning* and *applications of distributed learning* to real world problems [Park and Kargupta, 2002b]. We discuss each of them in what follows.

1.4.1.1 Distributed Learning Algorithms

Most of the distributed learning algorithms in the literature deal with homogeneous data. Among these, most of the existent algorithms work for horizontal data distributions, with a few exceptions that will be pointed out below.

Many of the approaches to distributed learning come from the desire to scale up algorithms to large data sets [Provost and Kolluri, 1999; Provost, 2000]. Conceptually there is a big difference between approaches to distributed learning coming from scaling up algorithms, where the data are distributed by the algorithm in order to increase the overall efficiency, and approaches that assume that data are inherently distributed and autonomous, and thus restrictions and constrains may need to be taken into account. The work in this dissertation falls in the second category. We say "learning from distributed data" as opposed to "distributed learning" to point out this difference.

Parallel Data Mining

Early work on distributed data mining appeared as a need to scale up learning algorithms to large data set [Provost and Kolluri, 1999]. Among other approaches to the problem of learning from large data sets, high performance parallel computing (a.k.a. parallel data mining) distinguishes itself as very useful for distributed settings as well.

Srivastava *et al.* [1999] proposed methods for distributing a large centralized data set to multiple processors to exploit parallel processing to speed up learning. Provost and Kolluri [1999] and Grossman and Guo [2001] surveyed several methods that exploit parallel processing for scaling up data mining algorithms to work with large data sets.

There has been a lot of research focused on parallelizing specific algorithms. For example, in [Amado *et al.*, 2003; Andrade *et al.*, 2003; Jin and Agrawal, 2003] the authors showed how the decision tree algorithm can be parallelized. In [Dhillon and Modha, 1999; Foti *et al.*, 2000; Samatova *et al.*, 2002] parallel clustering algorithms were considered. In [Tveit and Engum, 2003; Poulet, 2003] the authors proposed parallel solutions to some SVM algorithm variants. A lot of work [Agrawal and Shafer, 1996; Manning and Keane, 2001] has focused on parallelizing association rules [Agrawal and Shafer, 1996; Manning and Keane, 2001; Park *et al.*, 1995; Parthasarathy *et al.*, 2001; Wolff *et al.*, 2003; Zaiane *et al.*, 2001; Zaki, 1999].

Ensembles Approach to Distributed Learning

Several distributed learning algorithms have their roots in ensemble methods [Dietterich, 2000]. Thus, Domingos [1997] and Prodromidis *et al.* [2000] used ensemble of classifiers approaches to learning from horizontally fragmented distributed data, which involves learning separate classifiers from each data set and combining them typically using a weighted voting scheme. In general, this combination requires gathering a subset of data from each of the data sources at a central site to determine the weights to be assigned to the individual hypotheses (or alternatively shipping the ensemble of classifiers and associated weights to the individual data sources where they can be executed on local data to set the weights), which is not desirable. Other ensemble approaches were proposed in [Fern and Brodley, 2003; Hall and Bowyer, 2003; Jouve and Nicoloyannis, 2003; Tsoumakas and Vlahavas, 2002]. Besides the need to transmit some subset of data to the central site, there is another potential drawback of the ensemble of classifiers approach to learning from distributed data, mainly that the resulting ensemble of classifiers is typically much harder to comprehend than a single classifier. Another important limitation of the ensemble classifier approach to learning from distributed data is the lack of guarantees concerning generalization accuracy of the resulting hypothesis relative to the hypothesis obtained in the centralized setting.

Cooperation-based Distributed Learning

Although learning with cooperation scenarios could be very often met in real world situations, there are not many distributed learning algorithms that use the cooperation in an active way to obtain the final result, with a few notable exceptions.

Provost and Hennessy [1996] proposed a powerful, yet practical distributed rule learning (DRL) algorithm using cooperation. They make use of several criteria to estimate the probability that a rule is correct (and in particular to evaluate a rule), and define what it means for a rule to be satisfactory or acceptable over a set of examples (a rule can be acceptable for a local learner but not satisfactory for the batch learner). The algorithm tries to find acceptable local rules that are also satisfactory as global rules. In [Leckie and Kotagiri, 2002] the authors proposed an algorithm for learning to share distributed probabilistic beliefs. Morinaga *et al.* described another approach to collaborative data mining.

As opposed to collaboration by exchanging models (e.g. rules) between learners, in [Turinsky and Grossman, 2000] data could be moved from one site to another in order to fully exploit the resources of the network. One practical example of a learning algorithm that uses cooperation to exchange data is described in [Kargupta *et al.*, 1999] (this approach works for vertically distributed data as it will be described below).

Learning from Vertically Distributed Data

Although most of the distributed learning algorithms assume horizontal data fragmentation, there are a few notable exceptions. Bhatnagar and Srinivasan [1997] proposed algorithms for learning decision tree classifiers from vertically fragmented distributed data. WoRLD system [Aronis *et al.*, 1996] is a collaborative approach to concept learning from vertically fragmented data. It works by computing the cardinal distribution of feature values in the individual data sets, followed by propagation of this distribution across different sites. Features with strong correlations to the concept to be learned are identified based on the first order statistical approximation to the cardinal distribution. Being based on first order approximations, this approach is impractical for problems where higher order statistics are needed.

Tumer and Ghosh [2000] proposed an ensemble approach to combine local classifiers. They used an order statistics-based technique for combining high variance models generated from heterogeneous sites.

Park and his colleagues [Park and Kargupta, 2002a] observed that inter-site patterns cannot be captured by aggregating heterogeneous classifiers. To deal with this problem, at each site, they construct a subset of the data that a the particular classifier cannot classify with

high confidence and ship such subsets of data at the central site, where a classifier is build. This classifier is used when data at one site is classified with low confidence by the classifier at that site. Although this approach gives better results than simply aggregating the classifiers, it requires data shipping and its performance is sensitive to the sample size.

Kargupta and his group proposed a framework to address the problem of learning from heterogeneous data, called Collective Data Mining (CDM) [Kargupta *et al.*, 1999]. Given a set of labeled data, CDM learns a function that approximates it. CDM relies on the observation that any function can be represented in a distributed fashion using an appropriate set of basis functions. Thus, at each data source, the learner estimates the Fourier coefficients from the local data, and transmits them to a central site. These estimates are combined to obtain a set of Fourier coefficients for the function to be learned (a process which may require a subset of the data from each source to be transmitted to the central site). At present, there are no guarantees concerning the performance of the hypothesis obtained in the distributed setting relative to that obtained in the centralized setting. Furthermore, a given set of Fourier coefficients can correspond to multiple hypothesis.

Based on CDM framework, Kargupta *et al.* [1999] described an algorithm for learning decision trees from vertically fragmented distributed data using a technique proposed by Mansour [1994] for approximating a decision tree using Fourier coefficients corresponding to attribute combinations whose size is at most logarithmic in the number of nodes in the tree. The CDM framework is also used to design distributed clustering algorithms based on collective principal component analysis [Kargupta *et al.*, 2001] or to designed distributed algorithms for Bayesian network learning (structure or parameters) [Chen *et al.*, 2001; Chen and Krishnamoorthy, 2002; Chen *et al.*, 2003b; Sivakumar *et al.*, 2003].

Relational Learning

The task of learning from relational data has received significant attention in the literature in the last few years. One of the first approaches to relational learning was based on Inductive Logic Programming (ILP) [Muggleton, 1992]. Inductive Logic Programming is a broad field which evolved from the development of algorithms for the synthesis of logic programs from examples and background knowledge to the development of algorithms for classification, re-

gression, clustering, and association analysis [Dzeroski and Lavrac, 2001]. Due to its flexible and expressive way of representing background knowledge and examples, the field considers not only single-table representations of the data but also multiple-table representations, which makes it a good candidate for relational learning [Blockeel and Raedt, 1997]. However, the ILP techniques are limited in their capability to work with relational databases. Attempts to link ILP techniques with relational databases have been made in [Lindner and Morik, 1995; Blockeel and Raedt, 1997].

Knobbe *et al.* [1999] outlined a general framework for multi-relational data mining which exploits structured query language (SQL) to gather the information needed for constructing classifiers (e.g., decision trees) from multi-relational data. Based on this framework, multi-relational decision tree learning algorithms have been developed [Leiva *et al.*, 2002; Atramentov *et al.*, 2003] .

Probabilistic models, especially Bayesian Networks (BN) [Pearl, 2000], are similar to ILP approaches, but specify a probability distribution over a fixed set of random variables. Several approaches for combining first order logic and Bayesian Networks have been proposed in the literature. The most representative ones are Probabilistic Logic Programs (PLP) [Ngo and Haddawy, 1997], Relational Bayesian Networks (RBN) [Jaeger, 1997], and Probabilistic Relational Models (PRM) [Koller, 1999; Getoor *et al.*, 2001; Friedman *et al.*, 1999]. In spite of their different backgrounds, they all seem to share the commonalities represented by Bayesian Logic Programs (BLP) as shown in [Kersting and De Raedt, 2000].

Approaches for mining structural data in form of graph have been also proposed in [Cook and Holder, 2000; Gonzalez *et al.*, 2002]. In this framework, objects in the data correspond to vertices in the graph, and relationships between objects correspond to directed or undirected edges in the graph. A search for patterns embedded in graphs is performed. Once a pattern (substructure) is found, it is added to the graph in order to simplify it, by replacing instances of the substructure with the substructure itself.

Privacy Preserving Distributed Data Mining

Several approaches to distributed data mining appeared from the need to preserve the privacy of the information that is mined [Lindell and Pinkas, 2002]. In such case summaries

of the data need to be used instead of raw data. Clifton *et al.* [2002] proposed a set of tools that can be used to learn from data while preserving the privacy. Du and Atallah [2001] designed ways to do privacy-preserving collaborative scientific computations.

Some work has focused on specific algorithms design in the presence of privacy constraints: Du and Zhan [2002] introduced an algorithm for building decision trees on private data; Kantarcioglu and Clifton [2002] and Schuster *et al.* [2004] dealt with privacy-preserving distributed mining of association rules from horizontally partitioned data, while Vaidya and Clifton [2002] proposed an algorithm that works when data are vertically partitioned; Kargupta *et al.* [2003] proposed an algorithm for computing correlations in a vertically distributed scenario while preserving privacy; Lin *et al.* [2003] and Merugu and Ghosh [2003] presented algorithms for privacy preserving clustering using EM mixture modeling and generative models, respectively, from horizontally distributed data, while Vaidya and Clifton [2003] proposed a K-Means clustering over vertically partitioned data.

1.4.1.2 Architectures and Systems for Distributed Learning

Agent-oriented software engineering [Jennings and Wooldridge, 2001; Honavar *et al.*, 1998; Weiß, 1998] offer an attractive approach to implementing modular and extensible distributed computing systems. Each data site has one or more associated agents that process the local data and communicate the results to the other agents or to a central supervising agent that controls the behavior of the local agents. Java Agents for Meta-Learning [Stolfo and others, 1997] (distributed agent-based data mining system that uses meta-learning technique), BODHI [Kargupta *et al.*, 1999] (hierarchical agent-based distributed system for collective data mining), PADMA [Kargupta *et al.*, 1997] (tool for document analysis that works on a distributed environment based on cooperative agents) systems follow the agent-based architecture approach.

Another approach to address scalable distributed data mining is based on clusters of high-performance workstations connected by a network link. Papyrus [Grossman *et al.*, 2000] is a system for mining distributed data sources on a local and wide area cluster and a super cluster scenario. It is designed to find optimal strategies for moving results or models or data over the

network. The architecture in [Ashrafi *et al.*, 2002] is similar to JAM, PADMA and Papyrus, except that data sources can be heterogeneous. XML technique is used for data translation.

Chattratichat *et al.* [1999] proposed Kensington architecture based on a distributed component environment. Components are located on different nodes on a generic network like Intranet or Internet. Kensington is divided into client (provides interactive creation of data mining tasks), application server (responsible for task coordination and data management) and third level servers (provide high performance data mining services). PaDDMAS [Rana *et al.*, 2000] is another component-based system similar to Kensington. As opposed to Kensington, PaDDMAS allows easy insertion of custom-based components. Each component has an interface and the connection of two components is allowed only if they have compatible interfaces.

Krishnaswamy *et al.* [2003] noted that distributed data mining has evolved towards embracing the paradigm of application service providers, which allows small organizations or individuals to access software on demand. They proposed an architecture that demonstrates how distributed data mining can be integrated in application service providers in an e-commerce environment. A user is billed based on estimated costs and response times. The architecture proposed is based on integrating client-server and agent technologies. Sarawagi and Nagaralu [2000] explored a similar idea.

The Knowledge grid [Cannataro *et al.*, 2001; Cannataro and Talia, 2003; Talia, 2003; Sunderam, 2003; Du and Agrawal, 2002] is a reference software architecture for geographically distributed parallel and distributed knowledge application applications. It is built on top of a computational grid that provides dependable, consistent, and pervasive access to high-end computational resources. The Knowledge Grid uses the basic grid services and defines a set of additional layers to implement the services of distributed knowledge discovery on world wide connected computers where each node can be a sequential or a parallel machine. The Knowledge Grid enables the collaboration of scientists that must mine data that are stored in different research centers as well as analysts that must use a knowledge management system that operates on several data warehouses located in the different company establishments [Chervenak *et al.*, 1999].

Discovery Net project [Curcin *et al.*, 2002; Guo, 2003] introduced the idea that complex applications can make use of Grid technologies only if an application specific layer is introduced. Thus, in the Discovery Net architecture, there exists a layer that provides support for constructing and publishing Knowledge Discovery Services.

1.4.1.3 Distributed Learning Real World Applications

Distributed data mining algorithms can be applied to problems in various real world domains, such as: network intrusion detection [Bala *et al.*, 2002; Kumar, 2003], credit fraud detection [Chan *et al.*, 1999], text classification [Kuengkrai and Jaruskulchai, 2002], chain store database of short transactions [Lin *et al.*, 2002], geoscientific data [Shek *et al.*, 1996], financial data mining from mobile devices [Kargupta *et al.*, 2002], sensor-network-based distributed databases [Bonnet *et al.*, 2001], car-health diagnostics analysis [Wirth *et al.*, 2001], etc.

1.4.2 Information Integration

Information integration is another problem related to the work presented in this dissertation. Davidson *et al.* [2001] and Eckman [2003] surveyed alternative approaches to data integration. Hull [1997] summarized theoretical work on data integration. Because of our focus on knowledge acquisition from autonomous, semantically heterogeneous distributed data sources, query-centric, federated approaches to data integration are of special interest. A federated approach lends itself much better to settings where it is desirable to postpone specification of user ontology and the mappings between data source specific ontologies and user ontology until when the user is ready to use the system. The choice of a query centric approach enables users the desired flexibility in querying data from multiple autonomous sources in ways that match their own context or application specific ontological commitments (whereas in a source centric approach, what the data from a source should mean to a user are determined by the source).

Early work on multi-database systems [Sheth and Larson, 1990; Barsalou and Gangopadhyay, 1992; Bright *et al.*, 1992] focused on relational or object-oriented database views for

integrated access to data from several relational databases. However, these efforts were not concerned with autonomous semantically heterogeneous data sources. More recent work [Tsai *et al.*, 2001] used ontologies to integrate domain specific data sources. Wiederhold and Genesereth [1997] proposed mediator programs to integrate heterogeneous data sources. Some efforts at building such mediators for information integration from multiple data repositories (including semi-structured and unstructured data) include the TSIMMIS project at Stanford University [Garcia-Molina *et al.*, 1997; Chang and Garcia-Molina, 1999] the SIMS project [Arens *et al.*, 1993] and the Ariadne project [Knoblock *et al.*, 2001] at the University of Southern California, the Hermes project at the University of Maryland [Lu *et al.*, 1995], Information Manifold, a system developed at ATT Bell labs for querying WWW documents [Levy, 1998], and NIMBLE – a commercial system based on research at the University of Washington [Draper *et al.*, 2001]. Several data integration projects have focused specifically on integration of biological data: The SRS (Sequence Retrieval System) [Etzold *et al.*, 2003] developed at the European Molecular Biology Laboratory and marketed by LION Bioscience, IBM's DiscoveryLink [Haas *et al.*, 2001], the TAMBIS project in UK [Stevens *et al.*, 2003], the Kleisli project [Chen *et al.*, 2003a] and its successor K2 [Tannen *et al.*, 2003] at the University of Pennsylvania

These efforts addressed, and to varying degrees, solved the following problems in data integration: design of query languages and rules for decomposing queries into sub queries and composing the answers to sub queries into answers to the initial query. In related work, Tomasic *et al.* [1998] proposed an approach to scaling up access to heterogeneous data sources. Haas *et al.* [1997] investigated optimization of queries across heterogeneous data sources. Rodriguez-Martinez and Roussoloulos [2000] proposed a code shipping approach to the design of an extensible middleware system for distributed data sources. Lambrecht *et al.* [1999] proposed a planning framework for gathering information from distributed sources.

However, each of the systems summarized above has several significant limitations. SRS provides flexible ways to navigate and aggregate information, but offers fairly limited facilities for querying, and semantically integrating information from diverse sources. DiscoveryLink goes a step further than SRS in that it includes an explicit data model, the relational model,

which allows users to perform SQL queries over remote sources. Kleisli does not include a model of the available data sources, but does offer a query language and a collection of wrappers (Kleisli drivers) for accessing biological data sources. The K2 system incorporates some of the ideas from Kleisli, but includes some features absent in Kleisli notably, data dictionaries (for information retrieval), and a complex value model of the data which allows data values to be constructed by arbitrarily nesting tuples, collections (sets, bags, lists) and variants. The TAMBIS system uses a description logic formalism for representing its ontology which facilitates subsumption reasoning. User queries in TAMBIS are formulated in terms of the TAMBIS ontology. However, the mapping between TAMBIS ontology and data sources is quite restrictive. It does not allow multiple sources for the same kind of data (e.g., the use of both Swiss-Prot and PIR as sources of protein data) and it does not allow users to impose their own ontologies on the data sources.

Few of the systems mentioned above take into account semantic relationships between values of attributes used to describe instances (e.g., taxonomies over attribute values) in individual data sources.

1.4.3 Learning Classifiers from Heterogeneous Data

The combination of information integration with distributed learning algorithms is still a relatively new idea and thus there has not been much research focused on that yet. The work in this dissertation exploits this combination. In what follows, we describe two previous attempts to combine information integration and distributed learning, followed by an overview of our approach in this context.

InfoGrid [Giannadakis *et al.*, 2003] is a flexible Grid system that developed on top of Kensington [Chattratichat *et al.*, 1999] in order to answer the needs of the scientific community. It provides data publishing and integration mechanisms for a large range of different scientific applications in a generic way, while allowing specific queries for individual application domains, as opposed to the common middleware systems where all users are supposed to use the same language. InfoGrid achieves this functionality by introducing a layer of Information Integration Services where the querying middleware supports language parameterization

allowing specific application areas to maintain their own querying model while enabling heterogeneous information resources to be queried effectively. InfoGrid does not change the learning algorithms, it only prepares the data that they need. Hence, once the data required by an algorithm is collected, it is passed to the learning algorithm.

The work in [McClean *et al.*, 2002] brings the information integration problem a step closer to the learning problem by providing a way for the user to pose statistical queries in the user ontology. Each data source has a specific ontology and meta-data that describes the ontology and the relationship with other ontologies in the system. The authors do not assume that a global ontology exists, as most integration systems do. However, they assume that there exist mappings between local data source ontologies and one or several global ontologies stored in an ontology server, as well as mappings between global ontologies. Thus, mappings from data source ontologies to the user ontology can be found using intermediary mappings between global ontologies. They are provided by a negotiation agent that computes them dynamically in an automated way by searching the meta-data in the system, making the problem of answering queries more flexible.

In related work, McClean *et al.* [2003] use the mappings found by the negotiation agent to answer aggregate queries over heterogeneous distributed databases in the presence of data inconsistencies or imprecise data (data specified at different levels of granularity) that are likely to appear in such distributed scenarios. Thus, after a global ontology is constructed dynamically by analyzing the meta-data that relates the heterogeneous databases, the aggregates are derived by minimization of the Kullback-Leibler information divergence using the EM (Expectation-Maximization) algorithm. Depending on the global ontology a user query can be assessed as answerable, partially answerable, or unanswerable in advance of computing the answer itself.

The focus of the proposed research is on learning classifiers from a set of heterogeneous autonomous distributed data sources. The autonomous nature of the data sources implies that the learner has little control over the manner in which the data are distributed among the different sources. The heterogeneous nature of the data opens up a new direction that links data mining and information integration.

Unlike the papers summarized above, our approach [Caragea *et al.*, 2004d] offers a general approach to the design of algorithms for learning from distributed data that is provably exact with respect to its centralized counterpart. Central to our approach is a clear separation of concerns between hypothesis construction and extraction of sufficient statistics from data. This separation makes it possible to explore the use of sophisticated techniques for query optimization that yield optimal plans for gathering sufficient statistics from distributed data sources under a specified set of constraints describing the query capabilities and operations permitted by the data sources (e.g., execution of user supplied procedures). The proposed approach also lends itself to adaptation to learning from heterogeneous distributed data sources when the ontologies associated with the individual data sources are different from each other [Caragea *et al.*, 2004b]. Thus, provided well-defined mappings between ontologies can be specified, the proposed approach to learning from distributed data can be extended to yield an approach to learning from heterogeneous distributed data of the sort encountered in many large scale scientific applications.

In terms of information integration, our approach proposes a clear separation between ontologies used for data integration (which are supplied by users) and the procedures that use ontologies to perform data integration. This allows users to replace ontologies used for data integration *on the fly*, making it attractive for data integration tasks that arise in exploratory data analysis wherein scientists might want to experiment with alternative ontologies.

1.5 Outline

The rest of the dissertation is organized as follows:

- Chapter 2: A brief introduction to machine learning systems is given together with ways to evaluate such systems. Several classical machine learning algorithms are presented. A careful look at these algorithms leads to the observation that only certain statistics about data are used in the process of generating the algorithm output, which in turn leads to a reformulation of a learning algorithm in terms of information extraction and

hypothesis generation. Sufficient statistics for the learning algorithms presented are identified.

- Chapter 3: The problem of learning from distributed data is formally defined and a general strategy, based on the decomposition of the algorithm into information extraction from distributed data and hypothesis generation, is proposed. We show how this strategy can be applied to transform the algorithms introduced in Chapter 2 into efficient algorithms for learning from distributed data. We also introduce a statistical query language for formulating and manipulating statistical queries involved in learning algorithms.

- Chapter 4: The approach used for learning from distributed data sources is extended to yield an approach to learning from semantically heterogeneous data sources. We formally define ontologies and show how we can extend data sources and statistical query operators with ontologies in order to get sound and complete answers to statistical queries. The problem of answering queries from partially specified data is also addressed and a solution is proposed.

- Chapter 5: A system for answering queries from distributed heterogeneous autonomous data sources is designed. At the core of this system there is a query answering engine, which receives queries, decomposes them into sub-queries corresponding to the distributed data sources, finds optimal plans for execution, executes the plans and composes the individual answers it gets back from the distributed data sources into a final answer to the initial query.

- Chapter 6: We give an overview of INDUS, a federated, query centric approach to learning classifiers from distributed data sources and present AirlDm, a collection of machine learning algorithms, which are data source independent by means of sufficient statistics and data source wrappers. We show how AirlDM can be combined with INDUS to obtain implementations for algorithms for learning from distributed heterogeneous autonomous data sources. A case study is presented in the end of the chapter.

- Chapter 7: We conclude with a summary, a list of contributions that this dissertation makes and several directions for future work.

2 LEARNING CLASSIFIERS FROM DATA

In this Chapter we define the problem of learning from data and describe five learning algorithms (Naive Bayes, Decision Tree Algorithm, Perceptron Algorithm, Support Vector Machines and k-Nearest Neighbors algorithm). We show that any learning algorithm can be decomposed into two components: an information extraction component in which sufficient statistics for learning are collected and a hypothesis generation component in which sufficient statistics are used to construct a hypothesis. For each of the algorithms described, we will identify the sufficient statistics for learning.

2.1 Machine Learning Systems

Machine Learning is a multidisciplinary field that brings together scientists from artificial intelligence, probability and statistics, computational complexity, information theory, etc. A key objective of *Machine Learning* is to design and analyze *algorithms* that are able to *improve* the *performance* at some *task* through *experience* [Mitchell, 1997].

Definition 2.1. A *machine learning system* is specified by several components:

- *Learner:* An algorithm or a computer program that is able to use the experience to improve the performance. Usually the learner have finite resources (e.g. time and memory), so it should be able to use them efficiently.
- *Task:* A description of the task that the learner is trying to accomplish (e.g., learn a concept, a function, a language, etc.).
- *Experience source:* Specification of the information that the learner uses to perform the learning. The experience can take various forms such as:

  - *Examples:* The learner is presented with labeled examples about a particular task. Sometimes we refer to examples as *instances*
  - *Queries:* The learner can pose queries about a task to a knowledgeable teacher.
  - *Experiments:* The learner is allowed to experience with the task and learn from the effects of its actions on the task.

- *Background knowledge:* The information that the learner has about the task before the learning process (e.g. "simple" answers are preferable over "complex" answers). This information may simplify the learning process.

- *Performance Criteria:* Measure the quality of the learning output in terms of accuracy, simplicity, efficiency etc.

Definition 2.2. Let $\mathcal{X}$ be a *sample space* from where the examples are drawn and let $\mathcal{D}$ be the set of all possible subsets of the sample space $\mathcal{X}$. In general, we assume that the examples are randomly chosen from an *unknown distribution*. A collection of examples $D \in \mathcal{D}$ is called a *data set* or a *data source*.

Definition 2.3. Let $\mathcal{C}$ be the space of all possible models that we may want to learn or approximate, and $\mathcal{H}$ the space of the models that a learner can draw on in order to construct approximations of the models in $\mathcal{C}$. Thus, a learning algorithm outputs an element $h \in \mathcal{H}$, called the *hypothesis* about the data.

Definition 2.4. A *classification task* is a task for which the learner is given experience in the form of labeled examples, and it is supposed to learn to classify new unlabeled examples. Thus, in a classification task, the data D typically consists of a set of *training examples* . Each *training example* $\mathbf{x}$ is described by a set of *attribute values* $< a_1, \cdots, a_n >$. The *class label* of an example can take any value from a finite set $C = \{c_1, \cdots, c_m\}$. Hence, $D = \{(\mathbf{x_1}, y_1), \cdots, (\mathbf{x_t}, y_t)\}$, where $y_i \in C$ for all $i \in \{1, \cdots, t\}$. In a classification task, the learned hypothesis $h \in H$ is called a *classifier* (e.g., a decision tree, a support vector machine, etc. or even the data in the case of lazy learning).

Note: In this dissertation, we will concentrate on classification tasks.

Definition 2.5. For a classification task, we say that a hypothesis h is *consistent* with a set of training examples if it correctly classifies all the examples in the set. The *classification error* (a.k.a. *sample error* or *empirical error*) of a hypothesis with respect to a set of examples is the fraction of examples in the set that are misclassified by h. The *true error* of a hypothesis h is the probability that the hypothesis h will misclassify an example randomly chosen according to the underlying distribution.

As the underlying distribution is unknown, we cannot measure the true error of a hypothesis, but we can measure the classification error on a data set. If this is a good estimate of the true error, we can get a good estimate for the probability of misclassifying new unlabeled examples.

Definition 2.6. We say that a learner L is *consistent* if it outputs a hypothesis which is consistent with the set of training examples.

Definition 2.7. If H is a hypothesis space that a learner L is called upon to learn and D is a training set for the learner L, then the most probable hypothesis $h \in H$ given the data D is called a *maximum a posteriori* (MAP) hypothesis. According to the Bayesian theory $h_{MAP} = \arg\max_{h \in H} P(D|h)P(h)$, where $P(h)$ is the prior probability of h and $P(D|h)$ (called *likelihood*) is the probability to observe the data D given the hypothesis h. If we assume that all the hypotheses $h \in H$ are equally likely a priori, then any hypothesis that maximizes $P(D|h)$ is called *maximum likelihood* (ML) hypothesis.

We are interested in finding maximum a posteriori hypotheses since they are *optimal* in the sense that no other hypothesis is more likely. The following theorem gives us conditions that ensure that a maximum a posteriori hypothesis is found.

Theorem 2.8. [Mitchell, 1997] *"Every consistent learner outputs a MAP hypothesis, if we assume a uniform prior probability distribution over H (i.e., $P(h_i) = P(h_j)$ for all i, j), and if we assume deterministic, noise-free training data (i.e., $P(D|h) = 1$ if D and h are consistent, and 0 otherwise)."*

The *Minimum Description Length* (MDL) principle [Rissanen, 1978] provides a way to implement Occam's razor ("Prefer the simplest hypothesis that fits the data."), thus making it possible to take the complexity of a hypothesis into account when choosing the optimal hypothesis. It achieves this by performing a trade off between the complexity of the hypothesis and the number of errors of the hypothesis. Shorter hypotheses that make a few errors are preferred to longer consistent hypotheses. This also ensures that the problem of over-fitting the data is avoided.

In the next section, we will formally define the problem of *learning from data* by referring back to the definitions introduced in this section.

2.2 Learning from Data

Definition 2.9. The problem of *learning from data* can be summarized as follows: Given a data set D, a hypothesis class H, and a performance criterion P, the learning algorithm L outputs a classifier $h \in H$ that optimizes P. If $D = \{(\mathbf{x_1}, y_1), \cdots, (\mathbf{x_t}, y_t)\}$, then the training examples $\mathbf{x_i}$ for $i = 1, n$ represent *inputs* to the classifier h, while the labels y_i for $i = 1, n$ represent *outputs* of the classifier h. The goal of learning is to produce a classifier that optimizes the performance criterion of minimizing some function of the classification error (on the training data) and the complexity of the classifier (e.g., MDL). Under appropriate assumptions, this is likely to result in a classifier h that assigns correct labels to new unlabeled instances.

Thus, a learning algorithm for a classification task consists of two components: a *learning component* when the hypothesis is learned from training examples and a *classification component* when the learned hypothesis is used to classify new test examples (see Figure 2.1).

The boundary that defines the division of labor between the learning and the classification components depends on the particular learning algorithm used. Some learning algorithms do most of the work in the training phase (*eager learning algorithms*) while others do most of the work during the classification phase (*lazy learning algorithms*).

While in the case of eager learning a hypothesis is constructed during the learning phase,

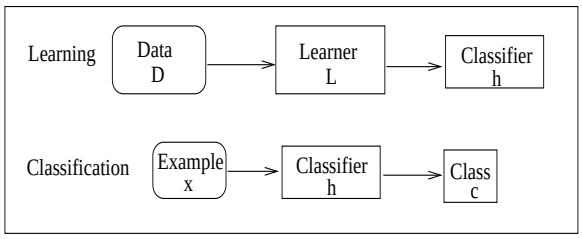

Figure 2.1 Learning algorithm (Up) Learning component (Down) Classification component

based on the training examples, in the case of lazy learning the training examples are simply stored and the generalization is postponed until a new instance needs to be classified. One advantage that lazy learning algorithms have over eager learning algorithms is that the target function is estimated locally (and thus it can be different for any new instance to be classified) as opposed to being estimated once for all the training examples. The main disadvantage is that the cost of classification in the case of lazy learning is higher than the cost of classification in the case of in eager learning, where most of the work is done once during the learning phase.

2.3 Examples of Algorithms for Learning from Data

In this section, we will describe a few popular eager learning algorithms (Naive Bayes Algorithm, Decision Tree Algorithm, Perceptron Algorithm, and Support Vector Machines) and also a well known lazy learning algorithm (k Nearest Neighbors).

2.3.1 Naive Bayes Classifiers

Naive Bayes is a highly practical learning algorithm [Mitchell, 1997], comparable to powerful algorithms such as decision trees or neural networks in terms of performance in some domains. In Naive Bayes framework, each example $\mathbf{x}$ is described by a conjunction of attribute values, i.e. $\mathbf{x} =< a_1, \cdots, a_n >$. The class label of an example can take any value from a finite set $C = \{c_1, \cdots, c_m\}$. We assume that the attribute values are conditionally independent given the class label. A training set of labeled examples,
$D = \{< \mathbf{x_1}, y_1 >, \cdots, < \mathbf{x_t}, y_t >\}$, is presented to the algorithm. During the learning phase, a hypothesis h is learned from the training set. During the evaluation phase, the learner is

asked to predict the classification of new instances $\mathbf{x}$.

If the new instance that needs to be classified is $\mathbf{x} =< a_1, \cdots, a_n >$, then according to Bayesian decision theory, the most probable class is given by

$$c_{MAP} = \arg\max_{c_j \in C} P(c_j | a_1, \cdots, a_n)$$

Using the Bayes theorem, we have:

$$c_{MAP}(\mathbf{x}) = \arg\max_{c_j \in C} \frac{P(a_1, \cdots, a_n | c_j) P(c_j)}{P(a_1, \cdots, a_n)} = \arg\max_{c_j \in C} P(a_1, \cdots, a_n | c_j) P(c_j)$$

Under the assumption that the attribute values are conditionally independent given the class label, the probability of observing the attribute values $a_1, \cdots, a_n$ given a class c_j is equal to the product of the probabilities for the individual attribute values for that class. Thus, $P(a_1, \cdots, a_n | c_j) = \prod_{i=1}^{n} p(a_i | c_j)$, which gives the following naive Bayes classification for the instance $\mathbf{x} =< a_1, \cdots, a_n >$:

$$c_{NB}(\mathbf{x}) = \arg\max_{c_j \in C} P(c_j) \prod_{i=1}^{n} P(a_i | c_j),$$

where the probabilities $P(c_j)$ and $P(a_i|c_j)$ can be estimated based on their frequencies over the training data. These estimates collectively specify the learned hypothesis h, which is used to classify new instances $\mathbf{x}$ according to the formula for $c_{NB}(\mathbf{x})$. The pseudocode for the Naive Bayes classifier is shown in Figure 2.2.

We mentioned before that the probabilities $P(c_j)$ and $P(a_i|c_j)$ are computed based on their frequencies in the training data. For example, for a class c, $P(c) = \frac{t_c}{t}$, where t_c is the number of training examples in class c and t is the total number of training examples. Although this estimate is good in general, it could be poor if t_c is very small. The Bayesian approach adopted in this case is to use a k-estimate (a.k.a. Laplace estimate) of the probability, defined as $\frac{t_c + kp}{t+k}$ [Mitchell, 1997]. Here p is a prior estimate of the probability we want to compute (e.g., $p = 1/m$ if there are m possible classes), and k is a constant called the equivalent sample size (it can be thought of as an augmentation of the set of t training examples by an additional

Naive Bayes Classifier

Learning Phase:
For each class c_j and each attribute value a_i compute the probabilities $P(c_j)$ and $P(a_i|c_j)$ based on their frequencies over the training data.

Classification Phase:
Given a new instance $\mathbf{x} =< a_1, \cdots, a_n >$ to be classified

Return $c_{NB}(\mathbf{x}) = \arg\max_{c_j \in C} P(c_j) \prod_{i=1}^{n} P(a_i|c_j)$

Figure 2.2 Naive Bayes classifier

k virtual examples distributed according to p).

We have seen that the Naive Bayes classifier relies on the assumption that the values of the attributes $a_1, \cdots, a_n$ are conditionally independent given the class value c. When this assumption is met, the output classifier is optimal. However, in general this assumption is not valid. Bayesian Networks [Pearl, 2000] relax this restrictive assumption by making conditional independence assumptions that apply to subsets of the variables. Thus, a Bayesian network [Pearl, 2000] models the probability distribution of a set of variables (attributes) by specifying a set of conditional independence assumptions and a set of conditional probabilities. Let $A_1, \cdots, A_n$ be random variables whose possible values are given by the sets $V(A_i)$, respectively. We define the *joint space* of the set of variables $A_1, \cdots, A_n$ to be the cross product $V(A_1) \times \cdots \times V(A_n)$, which means that each element in the joint space corresponds to one of the possible assignments of values to the variables $A_1, \cdots, A_n$. The probability distribution over these space is called *joint probability distribution*. A Bayesian Network describes the joint probability distribution for a set of variables. As in the case of Naive Bayes, each probability in the joint probability distribution can be estimated based on frequencies in the training data. Therefore, the results presented for Naive Bayes in the next chapters can be applied to Bayesian Networks as well.

2.3.2 Decision Tree Algorithm

Decision tree algorithms [Quinlan, 1986; Breiman *et al.*, 1984] are among some of the most widely used machine learning algorithms for building pattern classifiers from data. Their popularity is due in part to their ability to:

- select from all attributes used to describe the data, a subset of attributes that are relevant for classification;
- identify complex predictive relations among attributes; and
- produce classifiers that are easy to comprehend for humans.

The ID3 (Iterative Dichotomizer 3) algorithm proposed by Quinlan [Quinlan, 1986] and its more recent variants represent a widely used family of decision tree learning algorithms. The ID3 algorithm searches in a greedy fashion, for attributes that yield the maximum amount of information for determining the class membership of instances in a training set D of labeled instances. The result is a decision tree that correctly assigns each instance in D to its respective class. The construction of the decision tree is accomplished by recursively partitioning D into subsets based on values of the chosen attribute until each resulting subset has instances that belong to exactly one of the m classes. The selection of an attribute at each stage of construction of the decision tree maximizes the estimated expected information gained from knowing the value of the attribute in question.

Consider a set of instances D which is partitioned based on the class values $c_1, \cdots, c_m$ into m disjoint subsets $C_1, C_2, ..., C_m$ such that $D = \bigcup_{i=1}^{m} C_i$ and $C_i \bigcap C_j = \emptyset \; \forall i \neq j$. The probability that a randomly chosen instance $\mathbf{x} \in D$ belongs to the subset C_j is denoted by p_j. The *entropy* of a set D measures the expected information needed to identify the class membership of instances in D, and is defined as follows: $entropy(D) = -\sum_j p_j \cdot \log_2 p_j$. Given some impurity measure, the entropy [Quinlan, 1986] or Gini index [Breiman *et al.*, 1984], or any other measure that can be defined based on the probabilities p_j [Buja and Lee, 2001], we can define the *information gain* for an attribute a, relative to a collection of instances D as follows: $IGain(D, a) = I(D) - \sum_{v \in Values(a)} \frac{|D_v|}{|D|} I(D_v)$, where $Values(a)$ is the set of

all possible values for attribute a, D_v is the subset of D for which attribute a has value v, and $I(D)$ can be $entropy(D)$, Gini index, or any other suitable measure. As in the case of Naive Bayes, the probabilities p_j can be estimated based on frequencies in the training data, as follows: $p_j = \frac{|C_j|}{|D|}$, where we denote by $|\cdot|$ the cardinality of a set, and thus, the entropy can be estimated as follows: $entropy(D) = -\sum_j \frac{|C_j|}{|D|} \cdot \log_2 \left(\frac{|C_j|}{|D|}\right)$. The pseudocode of the algorithm is shown if Figure 2.3.

To keep things simple, we assume that all the attributes are discrete or categorical. However, this discussion can be easily generalized to continuous attributes by using techniques for discretizing the continuous-values attributes (e.g., by dividing the continuous interval where the attribute takes values into sub-intervals that correspond to discrete bins) [Fayyad and Irani, 1992; Witten and Frank, 1999].

Often, decision tree algorithms also include a pruning phase to alleviate the problem of over-fitting the training data [Mitchell, 1997; Esposito *et al.*, 1997]. For the sake of simplicity of exposition, we limit our discussion to decision tree construction without pruning. However, it is relatively straightforward to modify the algorithm to incorporate a variety of pruning methods.

2.3.3 Perceptron Algorithm

Let $D = \{(\mathbf{x_i}, y_i)|i = 1, t\}$ be a set of training examples, where $y_i \in C = \{0, 1\}$. We denote by $D^+ = \{(\mathbf{x_i}, y_i)|y_i = 1\}$, $D^- = \{(\mathbf{x_i}, y_i)|y_i = 0\}$ the sets of positive and negative examples, respectively. We assume that they are linearly separable, which means that there exists a linear discrimination function which has zero training error, as illustrated in Figure 2.4.

The learning task is to finding a vector, $\mathbf{w}^*$, called weight vector, such that:
$\forall \mathbf{x_i} \in D^+, \mathbf{w}^* \cdot \mathbf{x_i} > 0$ and $\forall \mathbf{x_i} \in D^-, \mathbf{w}^* \cdot \mathbf{x_i} < 0$. The perceptron algorithm [Rosenblatt, 1958] can be used for this purpose. Perceptrons are computing elements inspired by the biological neurons [McCulloch and Pitts, 1943; Minksy and Papert, 1969]. The pseudocode of the algorithm is presented in Figure 2.5.

Thus, we can see the perceptron weight vector as representing a separating hyperplane in the instance space. The perceptron outputs 1 if the instances lie on one side of the hyperplane

Decision Tree algorithm

Learning Phase

$ID3(D, A)$ (D set of training examples, A set of attributes).
Create a *Root* node for the tree.
if (all the examples in D are in the same class c_i)
{
    return (the single node tree *Root* with label c_i)
}
else
{
    Let $a \leftarrow BestAttribute(D)$
    for (each possible value v of a) **do**
    {
        Add a new tree branch below *Root* corresponding to the test $a = v$.
        if (D_v is empty)
        {
            Below this branch add a new leaf node with
            label equal to the most common class value in D.
        }
        else
        {
            Below this branch add the subtree $ID3(D_v, A - a)$.
        }
    }
}
return *Root*.
end-learning-phase

Classification Phase

Given a new instance **x**, use the decision tree having root *Root* to classify **x**, as follows:

- Start at the root node of the tree, testing the attribute specified by this node
- Move down the tree branch corresponding to the value of the attribute in the given example
- Repeat the process for the subtree rooted at the new node,

until this node is a leaf which provides the classification of the instance.

Figure 2.3 ID3 algorithm - greedy algorithm that grows the tree top-down, by selecting the best attribute at each step (according to the information gain). The growth of the tree stops when all the training examples are correctly classified

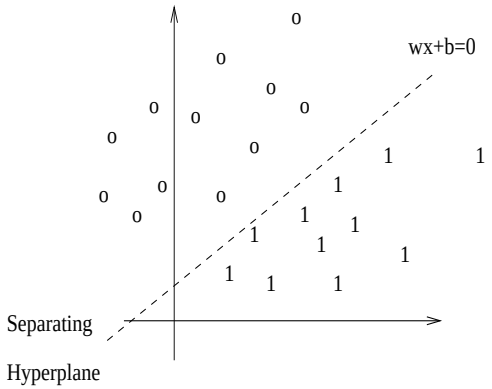

Figure 2.4 Linearly separable data set

and 0 if they lie on the other side. The intuition behind the update rule is to "step" in the direction that reduces the classification error. The value η, called *learning rate*, specifies the step size. The Perceptron Convergence Theorem [Minksy and Papert, 1969] guarantees that if the data are linearly separable the algorithm will find the separating hyperplane in a finite number of steps for any $\eta > 0$. The update rule of the Perceptron has the same mathematical form as the gradient descent rule, which is the basis for the Backpropagation [Rumelhart *et al.*, 1986] algorithm. The Backpropagation algorithm is in turn, the basis for many learning algorithms that search through spaces containing many types of hypotheses. Therefore, the discussion related to the Perceptron algorithm applies to a large class of neuron-based algorithms.

2.3.4 Support Vector Machines and Related Large Margin Classifiers

The Support Vector Machines (SVM) algorithm [Vapnik, 1998; Cortes and Vapnik, 1995; Scholkopf, 1997; Cristianini and Shawe-Taylor, 2000] is a binary classification algorithm. If the data are linearly separable, it outputs a separating hyperplane which maximizes the "margin" between classes. If data are not linearly separable, the algorithms works by (implicitly) mapping the data to a higher dimensional space (where the data become separable) and a maximum margin separating hyperplane is found in this space. This hyperplane in the high dimensional space corresponds to a nonlinear surface in the original space. Because they find a maximum margin separation, SVM classifiers are sometimes called "large margin

Perceptron Algorithm

Learning Phase

Initialize $w \leftarrow [0, \cdots, 0]$

do

{

1. **for** every example $(\mathbf{x_i}, y_i)$, compute $\mathbf{w} \cdot \mathbf{x_i}$. The output of the neuron is
$$o_i = \begin{cases} 1 & if \quad \mathbf{w} \cdot \mathbf{x}_i > 0 \\ 0 & if \quad \mathbf{w} \cdot \mathbf{x}_i \leq 0 \end{cases}$$
2. $\mathbf{w} \leftarrow \mathbf{w} + \eta(y_i - o_i)\mathbf{x}_i$

}

until(a complete pass through all the data sets results in no weight updates).

$\mathbf{w}^* \leftarrow \mathbf{w}$

Classification Phase

For a new instance $\mathbf{x}$

- assign $\mathbf{x}$ to the positive class if $\mathbf{x} \cdot \mathbf{w}^* > 0$;
- otherwise assign $\mathbf{x}$ to the negative class.

Figure 2.5 The Perceptron algorithm

classifiers". Large margin classifiers are very popular due to theoretical results that show that a large margin ensures a small generalization error bound [Vapnik, 1998] and also because they proved to be very effective in practice. As we will see below, SVM algorithm involves solving a quadratic optimization problem, which makes it inefficient for large data problems and difficult to implement. This is why algorithms that lead to the same solution as SVM, but are more efficient and easy to implement have received a lot of attention in recent years [Graepel and Herbrich, 2000; Zhang *et al.*, 2003a; Cesa-Bianchi *et al.*, 2001; Freund and Schapire, 1998; Friess *et al.*, 1998]. In what follows, we will describe the SVM algorithms and also two gradient-based algorithms that guarantee the same error bounds as SVM and are easier to use in practice.

2.3.4.1 Support Vector Machines

Let $D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_t, y_t)\}$, where $\mathbf{x}_i \in \mathcal{R}^n$ and $y_i \in \{-1, 1\}$ be a set of training examples for a 2-category classifier. Let $D^+ = \{\mathbf{x}_i | (\mathbf{x}_i, y_i) \in D \ \& \ y_i = +1\}$, and $D^- = \{\mathbf{x}_i | (\mathbf{x}_i, y_i) \in D \ \& \ y_i = -1\}$ be the set of positive and negative examples, respectively.

Suppose the training data are *linearly separable*. Then it is possible to find a hyperplane

h that partitions the n-dimensional pattern space into two half-spaces R^+ and R^- such that $D^+ \subset R^+$ and $D^- \subset R^-$. Each solution hyperplane can be specified by a pair $(\mathbf{w}, b)$ such that: $\mathbf{w} \cdot \mathbf{x}_i + b \geq 1 \ \ \forall \mathbf{x}_i \in D^+$, and $\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \ \ \forall \mathbf{x}_i \in D^-$. A solution hyperplane which satisfies the additional constraint $\min_{i=1,\cdots,t} |\mathbf{w} \cdot \mathbf{x}_i + b| = 1$ is called the *canonical hyperplane* and defines an one-to-one correspondence between the hyperplanes space and the set of pairs $(\mathbf{w}, b)$.

We call *margin* of the hyperplane h defined by a pair $(\mathbf{w}, b)$ with respect to a points $\mathbf{x}$ from the training set, the distance between the hyperplane h and the point $\mathbf{x}$, defined by: $d(\mathbf{x}, (\mathbf{w}, b)) = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|}$. Thus, the margin of a canonical hyperplane is equal to $\dfrac{1}{\|\mathbf{w}\|}$. It can be seen that the larger the margin of a hyperplane with respect to an example, i.e., the further away the example is from the discriminant, the easier to classify the example. Thus, we are interested in finding a "maximum margin" classifier that tries to maximize the distance between examples and the decision boundary as illustrated in Figure 2.6

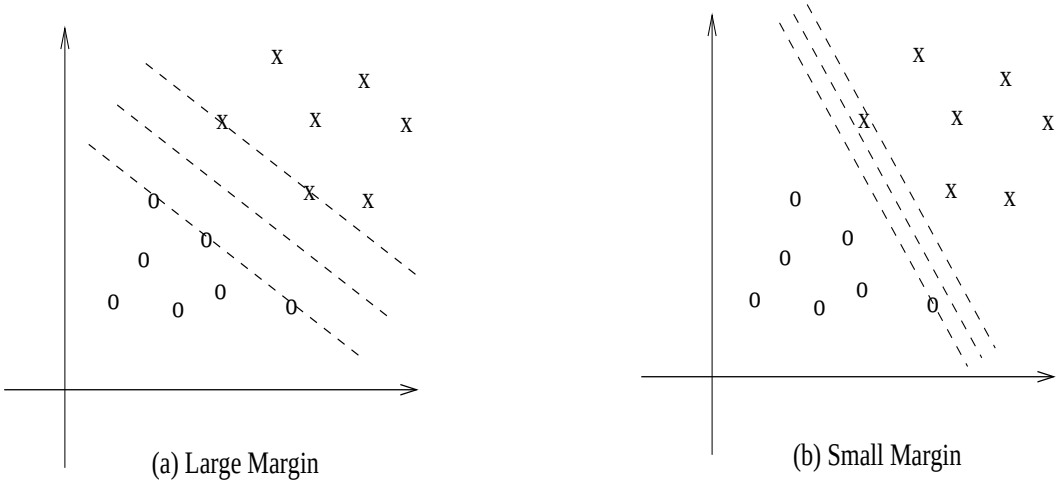

Figure 2.6 Maximum margin classifier

Among the hyperplanes that correctly classify the training set, SVM selects the one that minimizes $\|\mathbf{w}\|^2$, which involves solving the following quadratic programming problem:

$\min_{\mathbf{w},b} \Phi(\mathbf{w}) = \dfrac{1}{2}\|\mathbf{w}\|^2$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \ \ \forall i = 1, \cdots, t.$

The hyperplane which minimize $\|\mathbf{w}\|^2$ is the same as the hyperplane for which the margin of separation between the two classes, measured along a line perpendicular to the hyperplane,

is maximized. In order to solve the quadratic programming problem above, the dual problem (defined below) is considered and the technique of Lagrange multipliers is used. Thus, instead of solving the original problem, we solve the dual problem:

$$\max_{\lambda \geq 0} \theta(\lambda) = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \mathbf{x}_i^{'} \mathbf{x}_j$$

subject to $\sum_i \lambda_i y_i = 0$

The optimal solution will be $\mathbf{w}^* = \sum_{i=1}^{t} \lambda_i^* y_i \mathbf{x}_i$, where λ_i^*'s are the non-negative Lagrange multipliers corresponding to the constraints in the primal problem, and $b^* = y_i - \mathbf{w}^* \mathbf{x_i}$ for any $i = 1, t$ such that $\lambda_i^* > 0$. Thus the decision function can be written as $f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{t} y_i \lambda_i^* \mathbf{x} \cdot \mathbf{x_i} + b^*)$.

If the goal of the classification problem is to find a linear classifier for a non-separable training set, a new set of weights, called slack weights (measuring the extent to which the constraints are violated) can be introduced to define the following optimization problem:

$$\min_{\mathbf{x}, b, \xi} \Phi(\mathbf{w}, \xi) = \frac{1}{2} \|\mathbf{w}\|^2 \ + C(\sum_{i=1}^{i=t} \xi_i)^k$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i = 1, \cdots, t,$$

$$\xi_i \geq 0 \quad \forall i = 1, \cdots, t,$$

where C and k are proportional to the penalty for constraints violation. The decision function is similar to the one for the linearly separable case.

If the training examples are not linearly separable, the SVM algorithm works by mapping the training set into a higher dimensional *feature* space using an appropriate kernel function ϕ (Figure 2.7). The kernel function is chosen to ensure that the data become linearly separable in the feature space. Therefore the problem can be solved using linear decision surfaces in the higher dimensional space. Any consistent training set can be made separable with an appropriate choice of a feature space of a sufficiently high dimensionality [Vapnik, 1998]. However, in general, this can cause the learning algorithm to overfit the training data resulting in poor generalization. SVM avoids this problem by choosing the maximal margin hyperplane from the set of all separating hyperplanes [Vapnik, 1998]. The solution given by the SVM in this case will be of the following form:

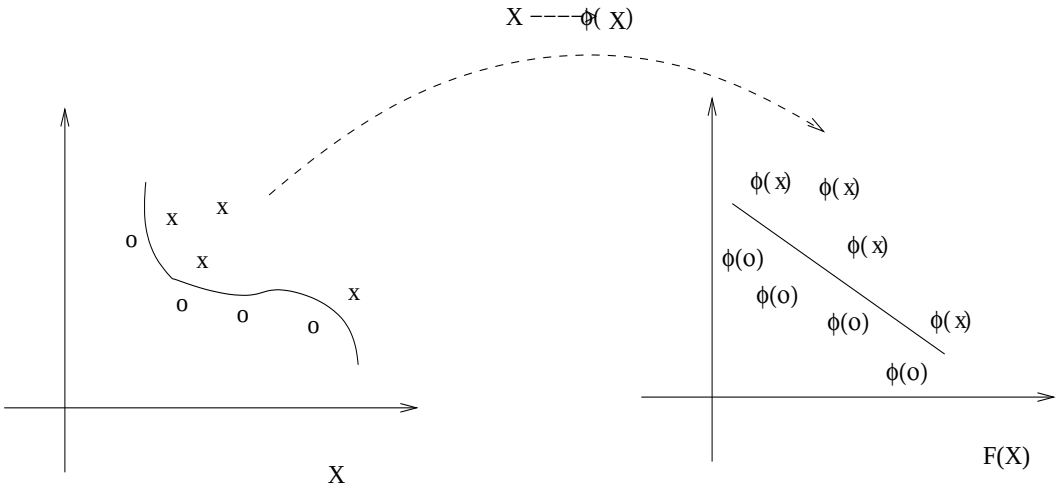

Figure 2.7 Non-linearly separable data mapped to a feature space where it becomes linearly separable

$f(\mathbf{x}) = \text{sign}(\mathbf{w}^* \cdot \phi(\mathbf{x}) + b^*) = \text{sign}(\sum_{i=1}^{t} y_i \lambda_i^* \phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i) + b^*)$

where $(\mathbf{w}^*, b^*)$ defines the solution hyperplane.

If we interpret λ_i^*'s as weights assigned to training instances $\mathbf{x}_i$'s, we can represent the maximum margin separating hyperplane as a weighted sum of the training patterns. In this weighted sum, the training patterns that lie far from this hyperplane receive weights of zero and only those patterns that lie close to the decision boundary between the two classes have non-zero weights. The training patterns that have non-zero weights are called the *support vectors*. The number of support vectors is usually a small fraction of the size of the training set. The pseudocode of the algorithm is shown in Figure 2.8.

Observation 2.10. It can be seen that the SVM algorithm uses the inner product between mapped data vectors $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$ for all i, j, instead of the data vectors themselves. Thus, we can learn SVM classifiers if we are given only the matrix $K = \phi(D)\phi(D)'$ instead of the data D. However, in order to classify new instances $\mathbf{x}$, the inner products between the instance $\mathbf{x}$ and the set of support vectors SV must be known as well.

Thus, as the *kernel matrix* is the object of interest, we characterize a kernel matrix using Mercer's Theorem 2.11 as follows:

Theorem 2.11. [Gradshteyn and Ryzhik, 1979] *A symmetric function* $K(\mathbf{x}, \mathbf{y})$ *can be ex-*

SVM Algorithm

Learning Phase

SVM(D:data, K:kernel)

Solve the optimization problem:

$$\max_{\lambda \geq 0} \theta(\lambda) = \sum_{i=1}^{t}(1-\xi_i)\lambda_i - \frac{1}{2}\sum_{i=1}^{t}\sum_{j=1}^{t}\lambda_i\lambda_j y_i y_j \cdot K(\mathbf{x}_i, \mathbf{x}_j) + C(\sum_{i=1}^{t}\xi_i)^k$$

subject to

$\sum_{i=1}^{t}\lambda_i y_i = 0$

$\xi_i \geq 0 \quad \forall i = 1, \cdots, t$

Let λ^* be the solution of this optimization problem.

Classification Phase

For a new instance $\mathbf{x}$

assign $\mathbf{x}$ to the class $f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{t} y_i\lambda_i^* \cdot K(\mathbf{x}, \mathbf{x}_i) + b^*)$

Figure 2.8 Support Vector Machines algorithm

pressed as an inner product $K(\mathbf{x},\mathbf{y}) = < \phi(\mathbf{x}), \phi(\mathbf{y}) >$ *for some* ϕ *if and only if* $K(\mathbf{x},\mathbf{y})$ *is positive semidefinite, i.e.* $\int K(\mathbf{x},\mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y}$ *for any* g*, or equivalently:*

$$\begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \cdots \\ K(x_2,x_1) & \ddots & \\ \vdots & & \end{bmatrix}$$

is positive semidefinite for any set $\{x_1, \cdots, x_n\}$*.*

Example 2.12. A common kernel is the Gaussian Kernel: $K(\mathbf{x},\mathbf{y}) = e^{\frac{1}{2}\|\mathbf{x}-\mathbf{y}\|}$

2.3.4.2 Sparse (Dual) Kernel Perceptrons

We have mentioned before that theoretical work [Vapnik, 1998] shows that a large margin is required for a classifier to ensure small generalization error bounds. However, the Occam's razor principle implies that "sparsity" of the solution vector (simplicity) is also important for generalization. Graepel and Herbrich [2000] show that there is a relation between margin and sparsity: "the existence of a large margin classifier implies the existence of sparse consistent classifiers in dual space". Furthermore, these classifiers can be found by the *dual*

perceptron algorithm. The classical perceptron algorithm [Rosenblatt, 1958] assumes that data are linearly separable. However, if this is not the case, as in the case of SVM, we could use a kernel function ϕ to map the data to a higher dimensional space where the data become linearly separable and learn the weight vector $\mathbf{w}$ in that space. In fact, Vapnik [1998] shows that it is better to learn in the dual representation, which means that we write the weight vector in terms of training instances, i.e. $w_\lambda = \sum_{i=1}^{|D|} \lambda_i \phi(\mathbf{x}_i)$, and learn the vector of coefficients $\lambda = (\lambda_0, \lambda_1, \cdots, \lambda_{|D|})$ instead of learning directly the components of $\mathbf{w}$. The dual perceptron algorithm is described in Figure 2.9. As opposed to the classical algorithm [Rosenblatt, 1958] where $\mathbf{w}_\lambda \leftarrow \mathbf{w}_\lambda + \eta y_i \phi(\mathbf{x}_i)$, the algorithm presented in Figure 2.9 adds a normalization term which reduces the upper bound on the number of iterations of the algorithm [Graepel and Herbrich, 2000].

Dual Perceptron Algorithm

Learning Phase

Learning rate: η.

Initialize $\lambda = (0, 0, \cdots, 0)$.

while (there exists an example $\mathbf{x}_i$ such that $y_i \cdot < \mathbf{w}_\lambda, \phi(\mathbf{x}_i) >_K \leq 0$) **do**

{

$$\lambda_i \leftarrow \lambda_i + \frac{\eta y_i}{\sqrt{K(\mathbf{x}_i, \mathbf{x}_j)}} \Leftrightarrow \mathbf{w}_\lambda \leftarrow \mathbf{w}_\lambda + \eta y_i \frac{\phi(\mathbf{x}_i)}{\|\phi(\mathbf{x}_i)\|_K}$$

}

Classification Phase

For a new instance $\mathbf{x}$

- assign $\mathbf{x}$ to the positive class if $\mathbf{x} \cdot \mathbf{w}^* > 0$;
- otherwise assign $\mathbf{x}$ to the negative class.

Figure 2.9 The Dual Perceptron algorithm

The main theorem in [Graepel and Herbrich, 2000] shows that "the *mere existence* of a large margin classifier λ^* is sufficient to guarantee a small generalization error for the solution λ of the dual perceptron although its attained margin is likely to be much smaller". This proves that the margin itself is not crucial for a small generalization error. However, the existence of a consistent large margin classifier implies the existence of a high sparsity classifier that can be efficiently found by the dual perceptron algorithm. It turns out that in practice, the error of the solution found by the dual perceptron is even smaller than the error found by SVM,

which makes this algorithm a simple and good candidate for problems where a large margin classifier exists.

2.3.4.3 Logistic Regression Approximation to SVM

Logistic regression (LR) is a traditional statistical tool [Darlington, 1990] that can be used to approximate SVM [Zhang *et al.*, 2003a]. When the data are linearly separable, LR models the conditional probability of the class label y given the instance $\mathbf{x}$:

$$p(y|\mathbf{x}) = \frac{1}{1 + \exp(-y(\mathbf{w}^T\mathbf{x} + b))},$$

where $\mathbf{w}$ and b define the separating hyperplane. The Regularized LR involves solving the following optimization problem:

$$\mathbf{w}^* = \arg\min_w \left\{ \frac{1}{n} \sum_{i=1}^{|D|} \log(1 + \exp(-y_i(\mathbf{w}^T\mathbf{x}_i + b))) + \lambda \mathbf{w}^T\mathbf{w} \right\},$$

whose Hessian matrix is positive definite. This means that the objective function of the regularized LR is strict convex, and thus it has a unique global solution [Luenberger, 1973].

Zhang *et al.* [2003] showed that a variant of the regularized logistic regression can be used to approximate SVM by defining a sequence of smooth functions that converge uniformly to the objective function of SVM. Thus, simple unconstrained optimization problems can be used to solve the SVM optimization problem.

2.3.5 k Nearest Neighbors Classifiers

The k nearest neighbors (k-NN) classifier [Cover and Hart, 1967; Mitchell, 1997] is a simple example of instance-based learning, also known as lazy learning. In the k-NN algorithm, the nearest neighbors are defined in terms of a metric (a.k.a. distance) $D(.,.)$ between instances. A metric is a function that satisfies the following properties for all $\mathbf{x}, \mathbf{y}, \mathbf{z}$ instances (i.e., vectors) [Frchet, 1906]:

- non-negativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$

- reflexivity: $D(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$
- symmetry: $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$
- triangle inequality: $D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z}) \geq D(\mathbf{x}, \mathbf{z})$

If we assume that all the instances are points in the n-dimensional space $\mathbf{R}^n$, let $\mathbf{x} =< a_1, \cdots, a_n >$ and $\mathbf{y} =< b_1, \cdots, b_n >$. It is easy to check that the following metrics in $\mathbf{R}^n$ satisfy the properties above:

- Euclidean distance: $d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$
- Minkowski metric: $d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n}|a_i - b_i|^k\right)^{\frac{1}{k}}$
- Manhattan distance: $d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n}|a_i - b_i|$

Let $C = \{c_1, \cdots, c_m\}$ be the set of class labels, and $D = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_t, \mathbf{y}_t)\}$ the set of training examples. Then the class label for a new instance $\mathbf{x}$ is given by the most common class label among the k training examples nearest to $\mathbf{x}$ (Figure 2.10).

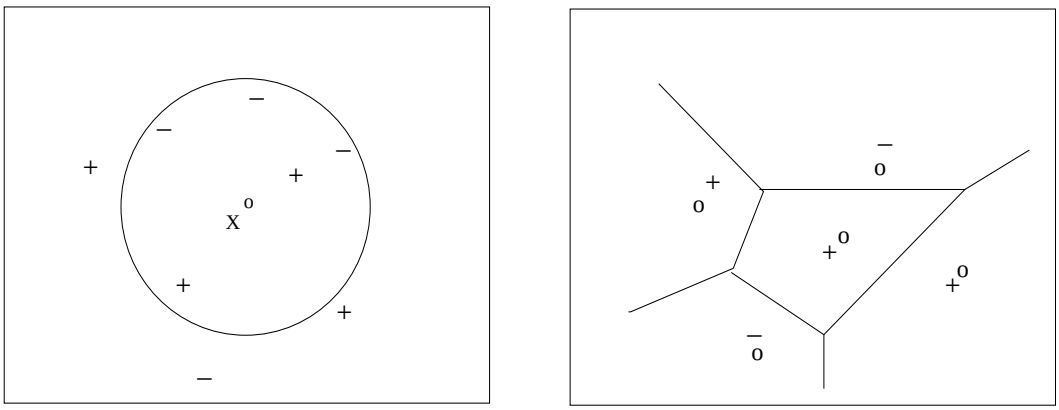

Figure 2.10 Decision boundary induced by the 1 nearest neighbor classifier

The pseudocode of the algorithm is shown in Figure 2.11. We can see that no general hypothesis h is learned. Instead, the algorithm computes the classification of a new instance $\mathbf{x}$ as needed. Alternatively, we can view this as assembling a hypothesis for each instance to be classified.

k-NN Algorithm

Learning Phase:
for (each training example $(\mathbf{x}_i, y_i)$)
{
add the example to the list *training_examples*
}

Classification Phase:
Given a new instance $\mathbf{x}$ to be classified:
let $\mathbf{x}_{i_1}, \cdots, \mathbf{x}_{i_k}$ be the k nearest neighbors of the instance $\mathbf{x}$ in the list *training_examples*
return

$$h(\mathbf{x}) = \arg\max_{c \in C} \sum_{j=1}^{k} \delta(c, y_{i_j}),$$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise.

Figure 2.11 The k Nearest Neighbors algorithm

2.4 Decomposition of Learning Algorithms into Information Extraction and Hypothesis Generation Components

The algorithms described in the previous section are representative for a large class of Machine Learning algorithms. We observe [Kearns, 1998; Caragea *et al.*, 2004d] that most of the learning algorithms use only certain *statistics* computed from the data D in the process of generating a hypothesis and classifying new instances. (Recall that a *statistic* is simply a function of the data. Examples of statistics include mean value of an attribute, counts of instances that have specified values for some subset of attributes, the most frequent value of an attribute, etc.) This yields a natural decomposition of a learning algorithm components (learning component and classification component) into two sub-components (see Figure 2.12): (1) an *information extraction* component that formulates and sends a *statistical query* to a data source and (2) a *hypothesis generation* component that uses the resulting statistic to modify a partially constructed algorithm output, sometimes represented also as a statistic (and further invokes the information extraction component if needed to generate the final algorithm output) [Caragea *et al.*, 2004d].

Definition 2.13. A *statistical query* $q(s)$ is any query q that returns a statistic s. One common type of query posed by many learning algorithms (Naive Bayes, Bayesian Networks,

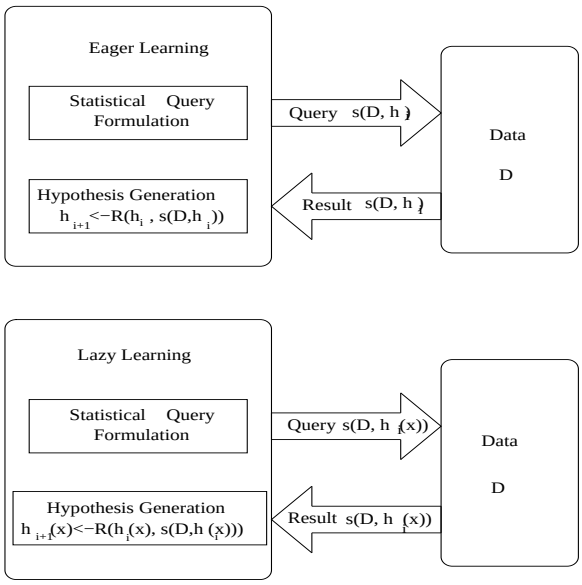

Figure 2.12 Learning revisited: identify sufficient statistics, gather the sufficient statistics and generate the current algorithm output

Decision Trees etc.) is called *count* or *joint count statistical query.*

Example 2.14. If D is data shown in Table 2.1, then the query that asks for the number of examples in D for which the class attributes $EnjoySport$ takes the value Yes is a count statistical query. The query that asks for the number of examples in D for which attribute $Humidity$ takes value Low given that the class attribute $EnjoySport$ takes value Yes, is a joint count statistical query.

In what follows, we formally define sufficient statistics for a learning algorithm [Caragea *et al.*, 2004d] and identify sufficient statistics for the algorithms described in Section 2.3 [Caragea *et al.*, 2000; 2001; 2004d].

2.5 Sufficient Statistics

Definition 2.15. [Casella and Berger, 2001] A statistic $s(D)$ is called a *sufficient statistic* for a parameter θ if $s(D)$ (loosely speaking) provides all the information needed for estimating the parameter θ from data D. Thus, sample mean is a sufficient statistic for mean of a Gaussian distribution.

Definition 2.16. [Casella and Berger, 2001] A sufficient statistic s for a parameter θ is called a *minimal sufficient statistic* if for every sufficient statistic s' for θ there exists a function g_s such that $g_s(s'(D)) = s(D)$.

We can generalize this notion of a sufficient statistic for a parameter θ to yield the notion of a sufficient statistic $s_L(D, \mathbf{x})$ for classifying a new instance x using a learning algorithm L applied to a data set D [Caragea *et al.*, 2004d]. Trivially, the data D is a sufficient statistic for classifying x using L applied to D. However, we are typically interested in statistics that are minimal or at the very least, substantially smaller in size than the whole data set D.

We observe that in the case of eager learning algorithms, a hypothesis is built during the learning phase of the algorithm, then it is used to classify new examples in the classification phase. Thus, the sufficient statistics for classifying a new example $\mathbf{x}$ are given by the example $\mathbf{x}$ itself and the statistics $s^l_L(D, h)$ that are needed to learn the hypothesis h from data D using the algorithm L. Hence, for eager learning algorithms, gathering sufficient statistics $s_L(D, \mathbf{x})$ reduced to gathering sufficient statistics $s^l_L(D, h)$ during the learning phase.

In the case of lazy learning algorithms, data are simply stored during the learning phase and a hypothesis $h(x)$ is build "on-the-fly" for each instance x during the classification phase. This means that the sufficient statistics $s_L(D, \mathbf{x})$ are given by the instance $\mathbf{x}$ itself and the statistics $s^c_L(D, h(\mathbf{x}))$ that are needed to learn $h(\mathbf{x})$ from D using the algorithm L. Hence, for lazy learning algorithms, gathering sufficient statistics $s_L(D, \mathbf{x})$ reduced to gathering sufficient statistics $s^c_L(D, h(\mathbf{x}))$ during the classification phase.

In some simple cases, it is possible to extract non trivial sufficient statistics $s^l_L(D, h)$ or $s^c_L(D, h(\mathbf{x}))$ in one step (e.g., when L is the standard algorithm for learning Naive Bayes or k-NN classifiers, respectively).

Definition 2.17. We say that $s^l_L(D, h)$ is a *sufficient statistic for learning h using the eager learning algorithm L* if there exists an algorithm that accepts $s^l_L(D, h)$ as input and outputs h.

Definition 2.18. We say that $s^c_L(D, h(\mathbf{x}))$ is a *sufficient statistic for learning $h(x)$ using the lazy learning algorithm L* if there exists an algorithm that accepts $s^c_L(D, h(\mathbf{x}))$ as input and

outputs $h(\mathbf{x})$.

In general, a hypothesis h or $h(\mathbf{x})$ is constructed by L by interleaving information extraction and hypothesis generation operations (see Figure 2.12). Thus, a decision tree learning algorithm would first obtain the sufficient statistics (expected information concerning the class membership of an instance associated with each of the attributes) for a single node decision tree (a partial hypothesis h_1), then follow up with queries for additional statistics needed to iteratively refine h_1 to obtain a succession of partial hypotheses $h_1, h_2, \cdots$ culminating in h, the final decision tree.

Observation 2.19. In what follows, with a little abuse of notation, we will denote by h either a hypothesis or a representation of the hypothesis (e.g., in the case of the SVM algorithm h could be the set of support vectors that determine the separating hyperplane and not the hyperplane itself).

Definition 2.20. We say that $s_L^l(D, h_i \rightarrow h_{i+1})$ is a sufficient statistic for the *refinement* of h_i into h_{i+1} if there exists an algorithm R^l which accepts h_i and $s_L^l(D, h_i \rightarrow h_{i+1})$ as inputs and outputs h_{i+1}.

Definition 2.21. We say that $s_L^c(D, h_i(\mathbf{x}) \rightarrow h_{i+1}(\mathbf{x}))$ is a sufficient statistic for the *refinement* of $h_i(\mathbf{x})$ into $h_{i+1}(\mathbf{x})$ if there exists an algorithm R^c which accepts $h_i(\mathbf{x})$ and $s_L^c(D, h_i(\mathbf{x}) \rightarrow h_{i+1}(\mathbf{x}))$ as inputs and outputs $h_{i+1}(\mathbf{x})$.

Definition 2.22. We say that $s_L^l(D, h_i \rightarrow h_{i+k})$ (where $k \geq 0$) is a sufficient statistic for *iteratively refining* h_i into h_{i+k} if h_{i+k} can be obtained through a sequence of refinements starting with h_i.

Definition 2.23. We say that $s_L^c(D, h_i(\mathbf{x}) \rightarrow h_{i+k}(\mathbf{x}))$ (where $k \geq 0$) is a sufficient statistic for *iteratively refining* $h_i(\mathbf{x})$ into $h_{i+k}(\mathbf{x})$ if $h_{i+k}(\mathbf{x})$ can be obtained through a sequence of refinements starting with $h_i(\mathbf{x})$.

Definition 2.24. We say that $s_L^l(D, (h_1, \cdots, h_m) \rightarrow h)$ is a sufficient statistic for the *composition* of $(h_1, \cdots, h_m)$ into h if there exists an algorithm C^l which accepts as inputs $h_1, \cdots, h_m$ and $s_l^L(D, (h_1, \cdots, h_m) \rightarrow h)$ and outputs the hypothesis h (or a representation of h).

Definition 2.25. We say that $s_L^c(D, (h_1(\mathbf{x}), \cdots, h_m(\mathbf{x})) \rightarrow h(\mathbf{x}))$ is a sufficient statistic for the *composition* of $(h_1(\mathbf{x}), \cdots, h_m(\mathbf{x}))$ into $h(\mathbf{x})$ if there exists an algorithm C^c which accepts as inputs $h_1(\mathbf{x}), \cdots, h_m(\mathbf{x})$ and $s_L^c(D, (h_1(\mathbf{x}), \cdots, h_m(\mathbf{x})) \rightarrow h(\mathbf{x}))$ and outputs the hypothesis $h(\mathbf{x})$ (or a representation of $h(\mathbf{x})$).

Definition 2.26. We say that $s_L^l(D, h)$ is a *sufficient statistic for learning the hypothesis h using the eager learning algorithm L and the training data D, starting with a null $h_0 = \phi$*, if h can be obtained from h_0 through some sequence of applications of composition and refinement operations.

Definition 2.27. We say that $s_L^c(D, h(\mathbf{x}))$ is a *sufficient statistic for learning the hypothesis $h(\mathbf{x})$ using the lazy learning algorithm L and the training data D, starting with a null $h_0(\mathbf{x}) = \phi$*, if $h(\mathbf{x})$ can be obtained from $h_0(\mathbf{x})$ through some sequence of applications of composition and refinement operations.

Definition 2.28. We say that $s_L(D, \mathbf{x})$ is a *sufficient statistic for classifying $\mathbf{x}$ using the eager learning algorithm L and the training data D* (or simply a *sufficient statistic for L* with respect to the learning phase of L) if there exist $s_L^l(D, h)$) such that $s_L(D, \mathbf{x})$ is obtained from $s_L^l(D, h)$ and $\mathbf{x}$.

Definition 2.29. We say that $s_L(D, \mathbf{x})$ is a *sufficient statistic for classifying $\mathbf{x}$ using the lazy learning algorithm L and the training data D* (or simply a *sufficient statistic for L* with respect to the classification phase of L) if there exist $s_L^c(D, h(\mathbf{x}))$) such that $s_L(D, \mathbf{x})$ is obtained from $s_L^c(D, h(\mathbf{x}))$ and $\mathbf{x}$.

Assuming that the relevant sufficient statistics (and the procedures for computing them) can be defined, finding the class of a new example $\mathbf{x}$ using an algorithm L and a data set D can be reduced to the computation of $s_L^l(D, h)$ (or $s_L^c(D, h(\mathbf{x}))$) through some sequence of applications of refinement and composition operations starting with the hypothesis $h_0 = \phi$ (or $h_0(\mathbf{x}) = \phi$).

2.6 Examples of Sufficient Statistics

2.6.1 Sufficient Statistics for Naive Bayes Classifiers

We have seen (Figure 2.2) that in the case of Naive Bayes classifiers a hypothesis (set of probabilities) is built during the learning phase. This hypothesis is used during the classification phase to classify new unseen instances. The set of probabilities $P(c_j)$ and $P(a_i|c_j)$, representing the hypothesis, can be computed based on counts of the form $t = count_D(x)$, $t_j = count_D(c_j)$, and $t_{ij} = count_D(a_i|c_j)$. Thus, these counts represent sufficient statistics for the hypothesis build during the learning phase of Naive Bayes classifiers [Caragea *et al.*, 2001]. They can be computed in one pass through the data as in Figure 2.13 where the data D is as in Table 2.1.

Table 2.1 Data set D: Decide *EnjoySport* based on *Weather* Data

Example	*Outlook*	*Wind*	*Humidity*	*EnjoySport*
1	Sunny	Strong	High	*No*
2	Sunny	Strong	Normal	*No*
3	Rainy	Weak	Normal	*Yes*

2.6.2 Sufficient Statistics for Decision Trees

The information requirements of the learning phase of ID3-like decision tree algorithms can be identified by analyzing the way in which the best attribute is chosen at each step of the algorithm (Figure 2.3). Different algorithms for decision tree induction differ from each other in terms of the criterion that is used to evaluate the splits that correspond to tests on different candidate attributes [Buja and Lee, 2001]. Many of the splitting criteria used in decision tree algorithms (e.g., information gain based on entropy or gini index) can be expressed in terms of relative frequencies computed from the relevant instances at each node. These relative frequencies can be reduced to counts of the form $t = count_{D|h}(x)$, $t_j = count_{D|h}(c_j)$, $t_{ij} = count_{D|h}(a_i|c_j)$ (where $D|h$ is the relevant data that satisfies the constraints specified by the partial decision tree h on the values of particular attributes). We

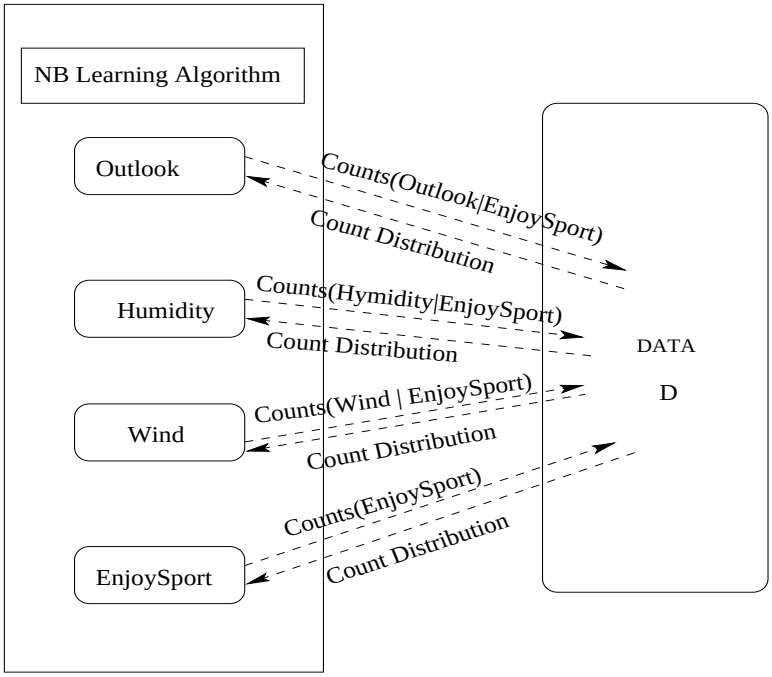

Figure 2.13 Naive Bayes classifiers learning as information extraction and hypothesis generation: the algorithm asks a joint count statistical query for each attribute in order to construct the classifier

notice that for a particular node h is actually the path to that node, which is a conjunction of attribute values. Thus, the sufficient statistics are joint counts that represent refinement sufficient statistics in the sense defined in Section 2.5 [Caragea *et al.*, 2004d]. They have to be obtained once for each node that is added to the tree starting with the root node (see Figure 2.14). A decision tree specific refinement operator uses these counts to compute the information gain for the possible splitting attributes and then it chooses the best attribute. The final decision tree constructed during the learning phase is used to classify new instances during the classification phase.

2.6.3 Sufficient Statistics for Perceptron Algorithm

The perceptron algorithm classifies new unseen examples based on the weight vector $\mathbf{w}$ computed during the training phase. By analyzing the algorithm in Figure 2.5, we notice that at each step, the weight $\mathbf{w}$ gets updated based on the current example $(\mathbf{x}_j, y_j) \in D$. As the weight $\mathbf{w}$ is computed directly from the data D, the value of the weight after one pass through the data (one iteration) can be seen as a minimal refinement sufficient statistic with

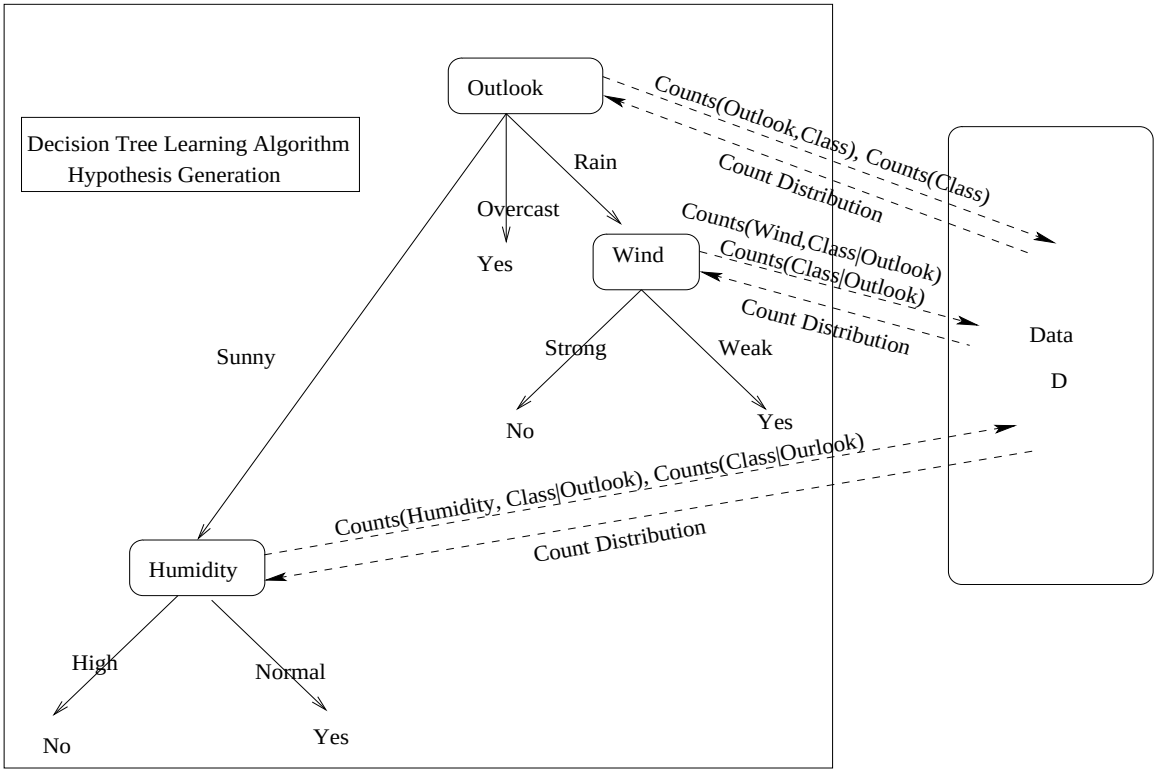

Figure 2.14 Decision Tree learning as information extraction and hypothesis generation: for each node, the algorithm asks a joint count statistical query and chooses the best attribute according to the count distribution

respect to the partial hypothesis constructed in one iteration i of the algorithm. We denote by $\mathbf{w}_{i+1}(D)$ the value of the weight computed from D at iteration $i+1$. Then, $s(D, \mathbf{w}_i(D))$ is a refinement sufficient statistic for $\mathbf{w}_{i+1}(D)$ (see Figure 2.15). The output of the algorithm is the final weight $\mathbf{w}(D)$.

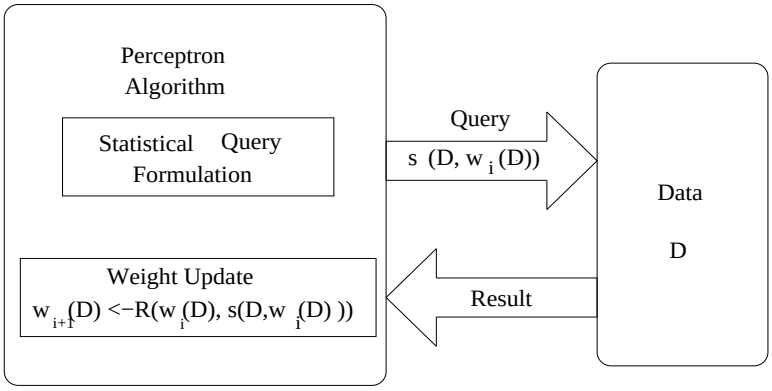

Figure 2.15 The Perceptron algorithm as information extraction and hypothesis generation: at each iteration $i+1$, the current weight $\mathbf{w}_{i+1}(D)$ is updated based on the refinement sufficient statistic $s(D, \mathbf{w}_i(D))$

2.6.4 Sufficient Statistics for SVM

The SVM algorithm (Figure 2.8) constructs a binary classifier that corresponds to a separating hyperplane that maximizes the margin of separation between instances belonging to two classes. The weight vector $\mathbf{w}$ that defines the maximal margin hyperplane is therefore a sufficient statistic for the SVM algorithm with respect to the learning phase. Because such a weight vector can be expressed as a weighted sum of a subset of training instances (called *support vectors*), the support vectors and the associated weights also constitute a sufficient statistic for SVM [Caragea *et al.*, 2000]. Thus, the algorithm can be decomposed into information extraction and hypothesis generation as in Figure 2.16. For Sparse Kernel Perceptrons and Logistic Regression Approximations to SVMs, the algorithms compute the weight incrementally by iterating through the data several times, as in the case of Perceptron algorithm. Thus, for these algorithms we have refinement sufficient statistics as in Figure 2.15.

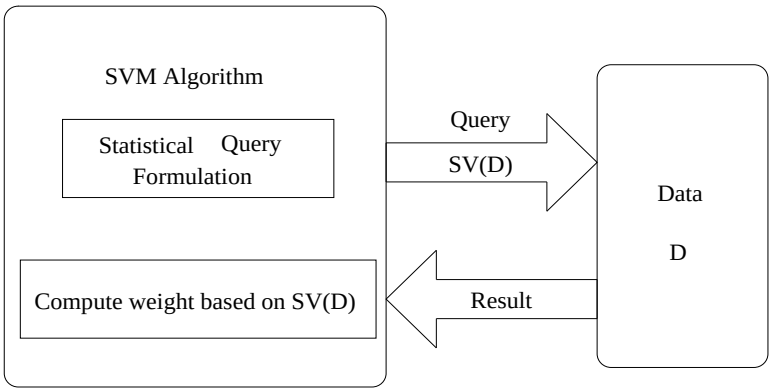

Figure 2.16 The SVM algorithm as information extraction and hypothesis generation: the algorithm asks for the support vectors and their associated weights) and the weight $\mathbf{w}$ is computed based on this information

2.6.5 Sufficient Statistics for k-NN

As can be seen in Figure 2.11, the learning phase of a k-NN algorithm consists simply of storing the data and the information extraction is done during the classification phase. Given a new example $\mathbf{x}$ to be classified the sufficient statistics with respect to $h(\mathbf{x})$ consist of the k nearest neighbors (training examples) of the new example $\mathbf{x}$ that needs to be classified. Given the k nearest neighbors, the class of the new example is determined by taking a majority vote among those examples, independent of the rest of the data. Furthermore, we notice that although the k nearest neighbors represent sufficient statistics for k-NN classifiers, they are not minimal sufficient statistics. The minimal sufficient statistics are given by the smallest k distances and the classes corresponding to them. As in the case of Naive Bayes, here the sufficient statistics can be computed in one step (see Figure 2.17).

2.7 Summary and Discussion

After introducing some background on machine learning systems and their evaluation, in this Chapter we defined the problem of learning from data and presented five classical learning algorithms (Naive Bayes, Decision Tree Algorithm, Perceptron Algorithm, Support Vector Machines and k-Nearest Neighbors algorithm) which are representative for a large class of learning problems.

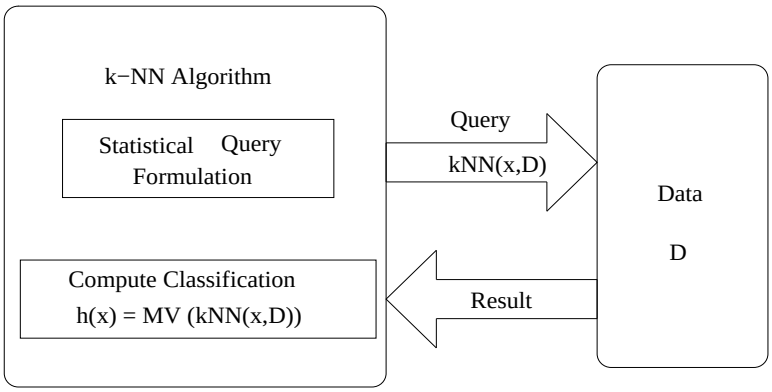

Figure 2.17 k-NN Algorithm as information extraction and hypothesis generation: for each example $\mathbf{x}$ the algorithm asks for the k nearest neighbors and computes the classification $h(\mathbf{x})$ taking a majority vote over these neighbors

In the context of learning in the presence of classification noise in the probabilistic learning model of Valiant and its variants [Valiant, 1984], Kearns [1998] formalized a related model of *learning from statistical queries* and showed that every class learnable in Valiant's model and its variants can also be learned in the new model and thus can be learned in the presence of noise (with one notable exception, the class of parity functions, which is not learnable from statistical queries, and for which no noise-tolerant algorithm is known). Intuitively, in the statistical query model, a learning algorithm is forbidden to examine individual examples of the unknown target function, but is given access to an oracle providing estimates of probabilities over the sample space of random examples. Thus, the example oracle $E(f, \mathcal{D})$ [Valiant, 1984] is replaced with a statistics oracle $STAT(f, \mathcal{D})$ [Kearns, 1998]. Every input to the statistics oracle is of the form (χ, α), where χ is any mapping of a labeled example to $\{0, 1\}$ and $\alpha \in [0, 1]$ and can be interpreted as the request for the probability $P_\chi = P_{x \in \mathcal{D}}[\chi(x, f(x)) = 1]$. As the oracle $STAT(f, \mathcal{D})$ will not return the exact value of P_χ, but only an approximation, α quantifies the amount of error the learning algorithm is willing to tolerate in this approximation.

Similar to [Kearns, 1998], we observed that most of the learning algorithms use only certain statistics computed from the data D in the process of generating hypotheses used to classify new instances [Caragea *et al.*, 2004d]. In the light of this observation, we revisited the classical definition of learning from data and showed that any learning algorithm can be decomposed

into two components: an information extraction component in which sufficient statistics for learning are collected and a hypothesis generation component in which sufficient statistics are used to construct a hypothesis [Caragea *et al.*, 2004d]. We defined formally the notion of sufficient statistics for classifying a new instance x using a learning algorithm L applied to a data set D and identified sufficient statistics for classifying instances using the algorithms of interest (Naive Bayes, Decision Tree Algorithm, Perceptron Algorithm, Support Vector Machines and k-Nearest Neighbors algorithm).

In the next Chapter, we will show how this new formulation of the problem of learning from data can be used to design algorithms for learning classifiers from distributed data.

3 LEARNING CLASSIFIERS FROM DISTRIBUTED DATA

In this chapter we will define the problem of *learning from distributed data* and show how we can transform learning algorithms as those presented in Chapter 2 into algorithms for learning from distributed data. We will also introduce criteria for comparing the two types of learning.

3.1 Learning from Distributed Data

Definition 3.1. In a distributed setting, the data are distributed across several data sources. Each data source contains only a fragment of the data. This leads to a fragmentation of a data set D. Two common types of *data fragmentation* are: *horizontal fragmentation* (Figure 3.1 (Left)), wherein (possibly overlapping) subsets of data tuples are stored at different sites; and *vertical fragmentation* (Figure 3.1 (Right)), wherein (possibly overlapping) sub-tuples of data tuples are stored at different sites. More generally, the data may be fragmented into a set

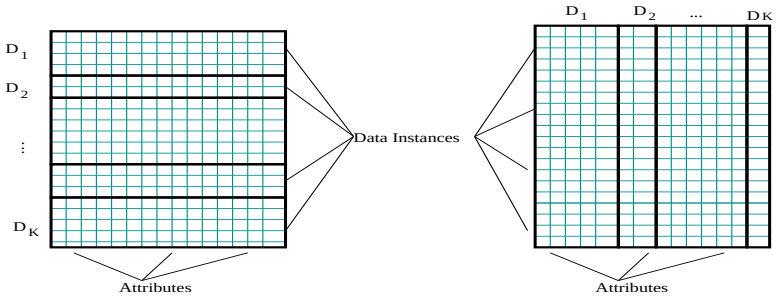

Figure 3.1 Data fragmentation: (Left) Horizontally fragmented data (Right) Vertically fragmented data

of *relations* (as in the case of tables of a relational database, but distributed across multiple sites) [Atramentov *et al.*, 2003] (see Figure 3.2).

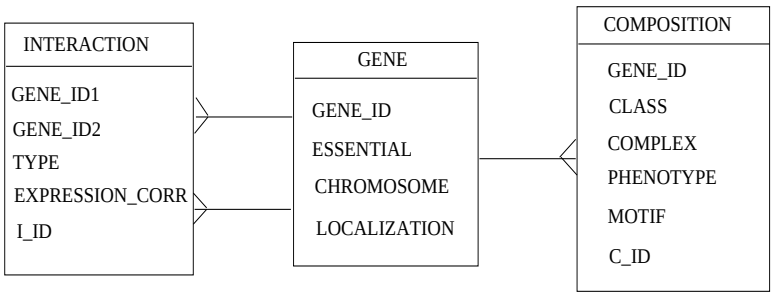

Figure 3.2 Multi relational database

If a data set D is distributed among the sites $1, \cdots, K$ containing data set fragments $D_1, \cdots, D_K$, we assume that the individual data sets $D_1, \cdots, D_K$ collectively contain all the information needed to construct the complete dataset D (at least in principle).

Thus, if the data D is horizontally distributed among the sites $1, \cdots, K$, then D can be reconstructed from $D_1, \cdots, D_K$ by simply taking the multi-set union of these subsets, i.e., $D = D_1 \cup \cdots \cup D_K$ (duplicates are allowed).

When the data are vertically distributed, we assume that each example has a unique index associated with it. Vertical fragments of the data are distributed across different sites. Each vertical fragment corresponds to a subset of the attributes that describe the complete data set. It is possible for some attributes to be shared (duplicated) across more than one vertical fragment, leading to overlap between the corresponding fragments. Let $A_1, A_2, \cdots, A_K$ indicate the set of attributes whose values are stored at sites $1, \cdots, K$ respectively, and let A denote the set of attributes that are used to describe the data tuples of the complete data set. Then in the case of vertically distributed data, we have: $A_1 \cup A_2 \cdots \cup A_K = A$. Let $D_1, D_2, \cdots, D_K$, denote the fragments of the dataset stored at sites $1, \cdots, K$ respectively, and let D denote the complete data set. Let the ith tuple in a data fragment D_j be denoted as $t^i_{D_j}$. Let $t^i_{D_j}.index$ denote the *unique index* associated with tuple $t^i_{D_j}$ and let $\times$ denote the *join* operation. Then the following properties hold: $D_1 \times D_2 \times \cdots \times D_K = D$ and $\forall\ D_j, D_k,\ t^i_{D_j}.index = t^i_{D_k}.index$. Thus, the sub-tuples from the vertical data fragments stored at different sites can be put together using their unique index to form the corresponding data tuples of the complete dataset. It is possible to envision scenarios in which a vertically fragmented data set might lack unique indices. In such a case, it might be necessary to use combinations of attribute values to infer

associations among tuples [Bhatnagar and Srinivasan, 1997]. In what follows, we will assume the existence of unique indices in vertically fragmented distributed data sets.

Definition 3.2. The distributed setting typically imposes a set of *constraints* Z on the learner that are absent in the centralized setting. For example:

- The constraints Z may prohibit the transfer of raw data from each of the sites to a central location while allowing the learner to obtain certain statistics from the individual sites (e.g., counts of instances that have specified values for some subset of attributes) or they may prohibit the execution of remote code to some of the data sources.
- The constraints Z may allow shipping raw data but they may specify that no sufficient statistics are provided (for example, operators for computing the statistics are not available).
- Sometimes it might be possible to ship code for gathering sufficient statistics that are not provided by a data source (for example, support vectors). However, the constraints Z may prohibit the execution of remote code at a data source.
- Some applications may impose a physical limit on the amount of information that can be shipped (e.g., no more than 100Mb per day).
- Other applications of data mining (e.g., knowledge discovery from clinical records), might impose constraints Z designed to preserve privacy.

Definition 3.3. The problem of *learning from distributed data* can be summarized as follows: Given the fragments $D_1, \cdots, D_K$ of a data set D distributed across the sites $1, \cdots, K$, a set of constraints Z, a hypothesis class H, and a performance criterion P, the task of the learner L_d is to output a hypothesis $h \in H$ that optimizes P using only operations allowed by Z. As in the case of centralized learning, this is likely to result in a *classifier* that can be used to classify new unlabeled data. Clearly, the problem of learning from a centralized data set D is a special case of learning from distributed data where $K = 1$ and $Z = \phi$.

Having defined the problem of learning from distributed data, we proceed to define some criteria that can be used to evaluate the quality of the hypothesis produced by an algorithm L_d for learning from distributed data relative to its centralized counterpart.

Definition 3.4. We say that an algorithm L_d for learning from distributed data sets $D_1, \cdots, D_K$ is *exact* relative to its centralized counterpart L if the hypothesis produced by L_d is identical to that produced by L from the complete data set D obtained by appropriately combining the data sets $D_1, \cdots, D_K$, and thus the classification error is the same.

Example 3.5. Let L_d be an algorithm for learning a Support Vector Machine (SVM) classifier $h_d : \Re^n \rightarrow \{-1, 1\}$, under constraints Z, from horizontally fragmented distributed data $D_1, \cdots, D_K$, where each $D_k \subseteq \Re^K \times \{-1, 1\}$. Let L be a centralized algorithm for learning an SVM classifier $h : \Re^K \rightarrow \{-1, 1\}$ from data $D \subseteq \Re^K \times \{-1, 1\}$. If $D = \cup_1^K D_k$, then we say that L_d is exact with respect to L if and only if $\forall X \in \Re^K, h(X) = h_d(X)$.

Observation 3.6. We should note that some learning algorithms involve making certain random choices, either in the learning phase or in the classification phase. For example, in the learning phase of the decision tree algorithm, if two possible splitting attributes have the same information gain, one of them is randomly chosen for splitting. Or in the classification phase of the Naive Bayes algorithm, if there are two equally probable classes for a test example $\mathbf{x}$, one of them is randomly chosen. Thus, when we define the *exactness* criterion for comparing learning from distributed data with learning from data, we assume that whenever a random choice needs to be made, both learning from distributed data and learning from data make the same choice.

Similar to the Definition 3.4, we can define exactness of learning from distributed data with respect to other criteria of interest (e.g., expected accuracy of the learned hypothesis). More generally, it might be useful to consider *approximate* learning from distributed data in similar settings. However, we focus on algorithms for learning from distributed data that are provably *exact* with respect to their centralized counterparts in the sense defined above, as proof of exactness of an algorithm for learning from distributed data relative to its centralized counterpart ensures that a large collection of existing theoretical (such as those introduced in

Chapter 2, Section 2.1) as well as empirical results obtained in the centralized setting apply in the distributed setting. Also questions addressed by the *Computational Learning Theory*, if answered in the centralized case, can be easily translated to the distributed case. Some examples of such questions are:

- What can we learn? What are the classes of hypothesis that are learnable by a learning algorithm? Which is the best algorithm for a specific problem? [Mitchell, 1997]
- When can we learn? What are the conditions under which learning is possible? How much training data is sufficient for learning? How much data do we need to ensure that the classification error is a good estimate for the true error? [Valiant, 1984; Vapnik and Chervonenkis, 1971]
- Have we learned? How *much* have we learned? Can we provide a high confidence bound on the true error of the learned hypothesis? [Langford, May 2002]

Goal: *Our goal is to design a strategy for transforming algorithms for learning from data into exact algorithms for learning from distributed data. We compare the resulting algorithms with the traditional algorithms in terms of time and communication complexity.*

3.2 General Strategy for Learning from Distributed Data

Our general strategy for designing an algorithm for learning from distributed data that is provably exact with respect to its centralized counterpart (in the sense defined above) follows from the observation made in Chapter 2, Section 2.4 that most of the learning algorithms use only certain *statistics* computed from the data D in the process of generating hypotheses and classifying new instances, and thus they can be decomposed in two components: (1) an information extraction component and (2) a hypothesis generation (information processing) component.

In light of this observation, the task of designing an algorithm L_d for learning from distributed data can be also decomposed into two components: (1) *information extraction from distributed data* and (2) *hypothesis generation*. The information extraction from distributed

data entails decomposing each statistical query q posed by the information extraction component of the learner into sub-queries $q_1, \cdots, q_K$ that can be answered by the individual data sources $D_1, \cdots, D_K$, respectively, and a procedure for combining the answers to the sub-queries into an answer for the original query q (see Figure 3.3). When the learner's access to data sources is subject to constraints Z the information extraction component has to be executable without violating the constraints Z. The transformation of the task of learning from distributed data into a sequence of applications of hypothesis refinement and hypothesis composition operations can be performed assuming serial or parallel access to the data sources $D_1, \cdots, D_K$ (see Figure 3.4).

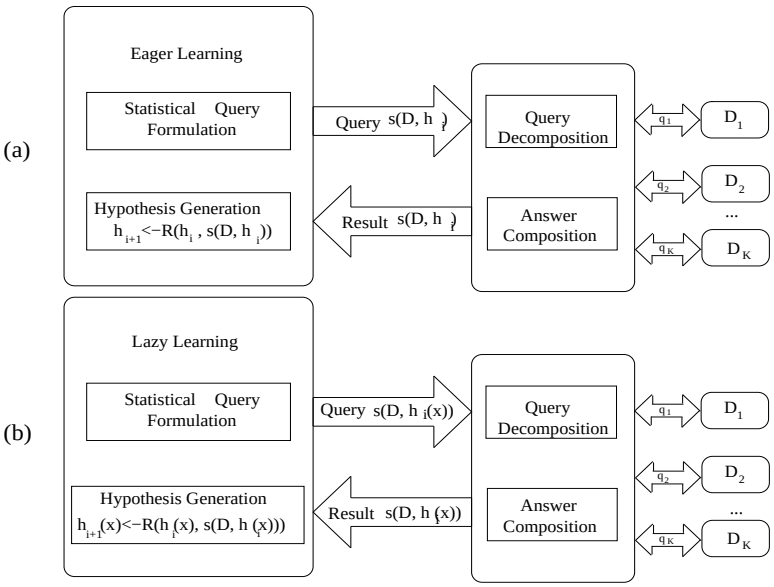

Figure 3.3 Exact distributed learning: distribute the statistical query among the distributed data sets and compose their answers.(a) Eager learning (b) Lazy learning

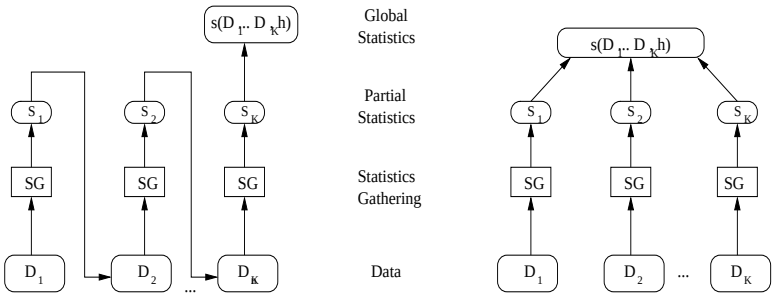

Figure 3.4 Distributed statistics gathering: (Left) Serial (Right) Parallel

The *exactness* of the algorithm L_d for learning from distributed data relative to its cen-

tralized counterpart, which requires access to the complete data set D, follows from the correctness (soundness) of the query decomposition and answer composition procedure. Thus, if D is distributed across the sites $D_1 \cup \cdots D_K$, let q be a statistical query posed by the information extraction component of the learner with respect to D and $q_1, \cdots, q_K$ sub-queries of q that can be answered by the individual data sources $D_1, \cdots, D_K$, respectively. We denote by C a procedure for combining the answers to the sub-queries into an answer for the original query q. Then, *a necessary and sufficient condition for an algorithm for learning from distributed data to be exact with respect to its centralized counterpart is the following:* $q(D) = C(q_1(D_1), \cdots, q_K(D_K))$.

We have mentioned that *count statistical queries* (Definition 2.13) are one common type of queries, whose answers are needed by many learning algorithms (Naive Bayes, Bayesian Networks, Decision Trees, etc.). The following lemma is true:

Lemma 3.7. *If D is horizontally distributed across the data sites $D_1, \cdots, D_K$ and q is a count statistical query over D that decomposes into sub-queries $q_1, \cdots, q_K$ with respect to $D_1, \cdots, D_K$, then $q(D) = C(q_1(D_1), \cdots, q_K(D_K))$, where C is the operation of adding up counts, i.e. $q(D) = q_1(D_1) + \cdots + q_K(D_K)$.*

Proof. Because D is horizontally distributed over $D_1, \cdots, D_K$, we have that $D = D_1 \cup \cdots \cup D_K$ and duplicates are allowed (see Definition 3.1), which means that we have exactly the same data in D and $D_1 \cup \cdots \cup D_K$. As the answers to count statistical queries are numbers, by adding up the numbers we obtain from the distributed data sets, we obtain the same number as if we answered the query q from D, i.e. $q(D) = q_1(D_1) + \cdots + q_K(D_K)$. □

Observation 3.8. Similar results can be obtained for different types of statistics, such as *weight refinement statistics*, needed for perceptron style algorithms or *distance-based statistics*, needed by k-NN style algorithms.

3.3 Algorithms for Learning Classifiers from Distributed Data

In this section we apply the strategy described above to design provably exact algorithms for learning from horizontally and vertically distributed data using Naive Bayes, Decision

Trees, Perceptron algorithm, Support Vector Machines and k-NN classifiers (see also [Caragea *et al.*, 2004d; 2001]). We show that although time complexity in the distributed case is similar to time complexity in the centralized case, the communication complexity can be sometimes drastically improved by performing learning from distributed data as opposed to centralized data.

We denote by $|D|$ the size of a data set D, $n = |A| = |x|$ the number of attributes used to describe an example x (except class), $v = |V|$ the maximum number of values of an attribute, $m = |C|$ the number of classes, and K the number of distributed data sites. Our complexity analysis will be done with respect to these numbers.

Definition 3.9. A *message* is defined as any unit of information (e.g., a number or a string) sent over the network. For the communication analysis, we compare the number of messages sent in the distributed as opposed to centralized case.

For the analysis that follows we make the following assumptions:

(1) Each local data source allows both shipping of the raw data and computation and shipping of the sufficient statistics. In general, this may not be the case. For example, some data sources may not allow data shipping or they may not allow the local execution of the operations necessary for computing the sufficient statistics, in which case the sufficient statistics can not be gathered and data has to be shipped. In such cases, the communication analysis changes accordingly.

(2) The data sources constraints do not change over time.

3.3.1 Learning Naive Bayes Classifiers from Distributed Data

We have seen in Chapter 2 that in the case of Naive Bayes classifiers counts of the form $t = count_D(x)$, $t_j = count_D(c_j)$, and $t_{ij} = count_D(a_i|c_j)$ represent sufficient statistics for the hypothesis constructed during the learning phase. They can be computed in one pass through the data. With these observations, we will show how we can gather the sufficient statistics for learning Naive Bayes classifiers from distributed data, and thus how we can construct Naive Bayes classifiers from horizontally and vertically fragmented data.

3.3.1.1 Horizontally Fragmented Distributed Data

When the data are horizontally distributed, each data source contains a subset of examples. Thus, all the attributes are present at each location and $D = D_1 \cup \cdots \cup D_K$, hence $|D| = |D_1| + \cdots + |D_K|$. In order to compute the counts over all the data, we need to compute $t^k = count_{D_k}(x)$, $t^k_j = count_{D_k}(c_j)$, $t^k_{ij} = count_{D_k}(a_i|c_j)$ at each location k and send them to a central location. The global counts are obtained at the central location by adding up local counts as follows: $t = \sum_{k=1}^{K} t^k$, $t_j = \sum_{k=1}^{K} t^k_j$, $t_{ij} = \sum_{k=1}^{K} t^k_{ij}$ (see Figure 3.5 for an example). The pseudocode for learning Naive Bayes classifiers from horizontally distributed

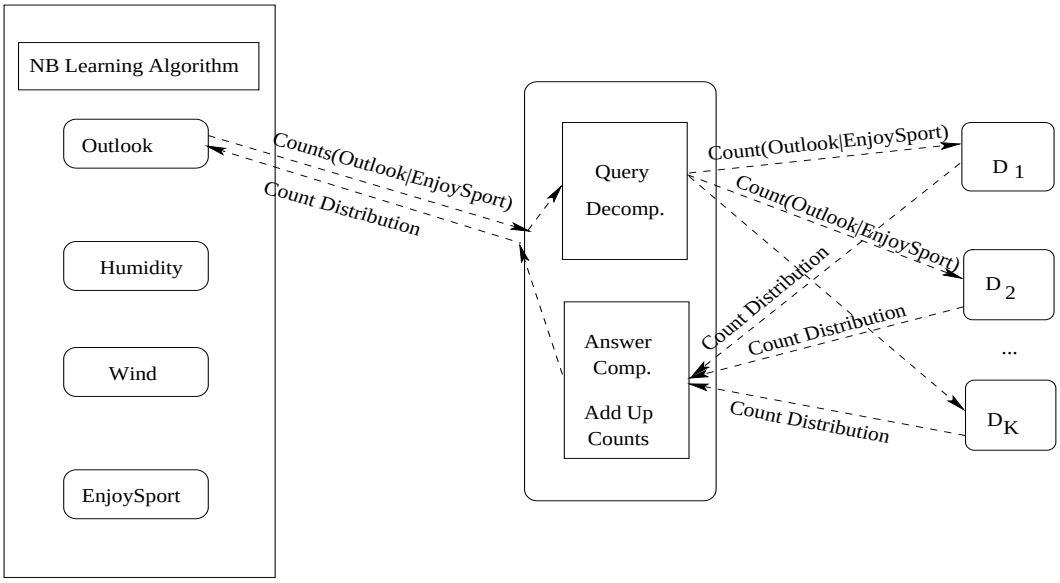

Figure 3.5 Learning Naive Bayes classifiers from horizontally distributed data: the algorithm asks a joint count statistical query for each attribute in order to construct the classifier. Each query is decomposed into sub-queries, which are sent to the distributed data sources and the answers to sub-queries are composed and sent back to the learning algorithm

data is shown in Figure 3.6.

Theorem 3.10. (Exactness) *The algorithm for learning Naive Bayes classifiers from horizontally distributed data, shown in Figure 3.6, is* exact *with respect to its batch counterpart, shown in Figure 2.2.*

Proof. We have seen that if the data are horizontally distributed, a subset of the training examples is stored at each location and thus all the attributes are present at each location.

Naive Bayes Classifier from Horizontally Fragmented Data

Learning Phase:

for (each data source D_k)
{
  Compute counts $t^k = count_{D_k}(x)$ and send them to the central location.
  for (each class c_j)
  {
    Compute the counts $t_j^k = count_{D_k}(c_j)$ and send them to the central location.
    for (each attribute value a_i)
    {
      Compute counts $t_{ij}^k = count_{D_k}(a_i|c_j)$ and send them to the central location.
    }
  }
}

At the central location, compute:

$$P(c_j) = \frac{\sum_{k=1}^{K} t_j^k}{\sum_{k=1}^{K} t^k},\ P(a_i|c_j) = \frac{\sum_{k=1}^{K} t_{ij}^k}{\sum_{k=1}^{K} t_j^k}.$$

Classification Phase:

Given a new instance $\mathbf{x} =< a_1, \cdots, a_n >$ to be classified,

Return $c_{NB}(\mathbf{x}) = \arg\max_{c_j \in C} P(c_j) \prod_{i=1}^{n} P(a_i|c_j)$

Figure 3.6 Naive Bayes classifier from horizontally fragmented data

Because the counts are additive, by computing the counts for an attribute locally, and adding up the local counts at the central place, we obtain the same numbers as if we brought all the data together and computed the counts globally (see Lemma 3.7). This means that we obtain the same statistics for both distributed and centralized data, therefore, the algorithm for learning from horizontally distributed data is exact. □

Theorem 3.11. (Time Complexity) *If both serial and parallel access to the data are allowed, then parallel access is preferred as it results in an algorithm for learning Naive Bayes classifiers from horizontally distributed data, shown in Figure 3.6, that is K times faster than the algorithm for learning Naive Bayes classifiers from centralized data, shown in Figure 2.2.*

Proof. We can see that for both algorithms (Figure 3.6 and Figure 2.2), the computation of the counts can be done with one pass through the data. However, in the distributed case, we can compute the counts at each location k independent of the counts at the other locations. Thus, if parallel data access is performed, the counts can be computed in parallel, which makes the algorithm for learning Naive Bayes classifiers from distributed data K times faster then the algorithm for learning from centralized data. Therefore, parallel access to data is preferred to serial access to data. □

Theorem 3.12. (Communication Complexity) *Under the assumption that both local computation of the sufficient statistics and shipping of the raw data are possible, then the algorithm for learning Naive Bayes classifiers from horizontally distributed data, shown in Figure 3.6, is preferable to the algorithm for learning Naive Bayes classifiers from centralized data, show in Figure 2.2 in terms of communication complexity if: $O(|V||C|K) < O(|D|)$.*

Proof. We first estimate the communication complexity in the distributed case: Each data source D_k computes t^k, t^k_j, t^k_{ij} for each class c_j and each attribute value a_i. The number of messages sent over the network is $(|A||V||C| + |C| + 1)K$. In the centralized case, all the data are shipped to the central location. Therefore, the number of messages sent is $(|D_1| + \cdots + |D_K|)(|A| + 1) = |D|(|A| + 1)$. Thus, learning of Naive Bayes classifiers from distributed data is preferred to learning from centralized data if $(|A||V||C| + |C| + 1)K \leq |D|(|A| + 1)$, which implies $O(|V||C|K) < O(|D|)$. □

Observation 3.13. The inequality $|V||C|K < |D|$ is usually satisfied in practice.

Example 3.14. Assume the following scenario which can be often met in practice: the data set D contains $|D| = 1,000,000$ examples which are distributed among K data sources. Each example is described by $|A| = 100$ attributes, and each attribute can take up to $|V| = 10$ possible values. An example belongs to one of $|C| = 10$ possible classes. Then the communication complexity in the distributed case is $(100 \cdot 10 \cdot 10 + 10 + 1)10 = 100,110$. The communication complexity in the centralized case is $1000000(100 + 1) = 101,000,000$ which is very large compared to the communication in the distributed case.

3.3.1.2 Vertically Fragmented Distributed Data

When the data are vertically fragmented, we have sub-tuples of all the examples at each site, which means that the data corresponding to an attribute is all at the same location. If we denote by A_k the set of attributes at each location k, then $A = \cup_{k=1}^{K} A_k$ and $|A| = \sum_{k=1}^{K} |A_k|$. In this case, we can compute $t = t_k = count_{D_k}(x)$ and $t_j = t_j^k = count_{D_k}(c_j)$ at any location k, and then compute $t_{ij} = t_{ij}^k = count_{D_k}(a_i|c_j)$ at the location k where the attribute a_i can be found. The pseudocode of the algorithm for learning from vertically distributed data is shown in Figure 3.7.

Theorem 3.15. (Exactness) *The algorithm for learning Naive Bayes classifiers from vertically distributed data, shown in Figure 3.7, is* exact *with respect to the algorithm for learning Naive Bayes classifiers from centralized data, shown in Figure 2.2.*

Proof. We mentioned that if the data are vertically distributed, then all the data related to an attribute is located at one site. Thus, by computing the counts for an attribute locally, we obtain the same numbers as if we brought all the data together and computed the counts globally, which means that we obtain the same statistics for both distributed and centralized learning, i.e. the algorithm for learning from distributed data is exact. □

Theorem 3.16. (Time Complexity) *If both serial and parallel access to the data are allowed, then parallel access is preferred as it results in an algorithm for learning Naive Bayes classifiers*

Naive Bayes Classifier from Vertically Fragmented Data

Learning Phase:

Let k be any location between 1 and K.
Compute $t = count_{X_k}(x)$ and send t to the central location.
for (each class c_j)
{
  Compute $t_j = count_{D_k}(c_j)$ and send t_j to the central location.
}
for (each data source D_k)
{
  for (each data class c_j and each attribute value a_i at site D_k)
  {
    Compute the counts $t_{ij} = count_{D_k}(a_i|c_j)$ based on the training data D_k.
    Send these counts to the central location.
  }
}
At the central location, compute:
$P(c_j) = \frac{t_j}{t}$, $P(a_i|c_j) = \frac{t_{ij}}{t_j}$.

Classification Phase:

Given a new instance $\mathbf{x} =< a_1, \cdots, a_n >$ to be classified,
Return $c_{NB}(\mathbf{x}) = \arg\max_{c_j \in C} P(c_j) \prod_{i=1}^{n} P(a_i|c_j)$

Figure 3.7 Naive Bayes classifier from vertically fragmented data

from vertically distributed data, shown in Figure 3.7, that is K times faster than the algorithm for learning Naive Bayes Classifiers from centralized data, shown in Figure 2.2

Proof. We can see that for both algorithms (Figure 3.7 and Figure 2.2) the computation of the counts can be done with one pass through the data. However, in the distributed case, we can compute the counts at each location k independent of the counts at the other locations. Thus, if parallel data access is performed, the counts can be computed in parallel, which makes the algorithm for learning from distributed data K times faster then the algorithm for learning from centralized data. Therefore, parallel access to data is preferred to serial access to data. □

Theorem 3.17. (Communication Complexity) *Under the assumption that both local computation of the sufficient statistics and shipping of the raw data are possible, the algorithm for learning from vertically fragmented data, shown in Figure 3.7, is preferable to algorithm for learning from centralized data, shown in Figure 2.2, in terms of communication complexity if:* $O(|A||V||C|) < O(|D|(|A| + K))$.

Proof. We first estimate the communication complexity of the algorithm for learning from distributed data: One of the data sources computes t and t_j for any $j = 1, m$ and sends them to the central location. Then, each data source D_k computes t_{ij} for each class c_j and those attribute values a_i located at the location k. The number of messages sent over the network is $((|A_1| + \cdots + |A_K|)|V||C| + |C| + 1) = (|A||V||C| + |C| + 1)$. In the centralized case, all the data are shipped to the central location. Therefore, the number of messages sent is $(|D|(|A_1|+2)+(|A_2|+1)\cdots+|D|(|A_K|+1)) = |D|(|A|+K+1)$ (just the first data source sends the class labels, the others send the attributes at their sites and the unique index that is used to put the data together). Thus, learning of Naive Bayes classifiers from vertically distributed data is preferred to learning from centralized data if $(|A||V||C| + |C| + 1) \leq |D|(|A| + K + 1)$, which implies $O(|A||V||C|) < O(|D|(|A| + K))$. □

Observation 3.18. The inequality $|A||V||C| < |D|(|A| + K)$ is usually satisfied in practice.

Example 3.19. We consider again a practical scenario: the data set D contains $|D| = 1,000,000$ examples. Each example is described by $|A| = 100$ attributes, each attribute can

take up to $|V| = 10$ possible values and the attributes are distributed among $K = 10$ data sources. An example belongs to one of $|C| = 10$ possible classes. Then the communication complexity in the distributed case is $100 \cdot 10 \cdot 10 + 10 + 1 = 10,011$. The communication complexity in the centralized case is $1000000(100 + 10 + 1) = 111,000,000$ which is very large compared to the communication in the distributed case.

3.3.2 Learning Decision Tree Classifiers from Distributed Data

We have shown in Chapter 2 that counts of the form $t = count_{D|h}(x)$, $t_j = count_{D|h}(c_j)$, $t_{ij} = count_{D|h}(a_i|c_j)$ (where $D|h$ is the relevant data that satisfies the constraints specified by the partial decision tree h on the values of the attributes on the current path) represent sufficient statistics for the refinement of the partial decision tree h. They have to be collected once for each node of the final tree starting with the root node. A decision tree specific refinement operator R_{DT} uses these counts to compute the information gain for the possible splitting attributes and it chooses the best attribute for splitting. We will show how we can gather the sufficient statistics for learning exact decision trees from horizontally and vertically fragmented data.

Assume that given a partially constructed decision tree, we want to choose the best attribute for the next split. Let $a_j(\pi)$ denote the attribute at the jth node along a path π starting from the attribute $a_1(\pi)$ that corresponds to the root of the decision tree, leading up to the node in question $a_l(\pi)$ at depth l. Let $v(a_j(\pi))$ denote the value of the attribute $a_j(\pi)$, corresponding to the jth node along the path π. For adding a node below $a_l(\pi)$, the set of examples being considered satisfies the following constraints on values of the attributes on the path π: $L(\pi) = [a_1(\pi) = v(a_1(\pi))] \wedge [a_2(\pi) = v(a_2(\pi))] \wedge \cdots \wedge [a_l(\pi) = v(a_l(\pi))]$, where $[a_j(\pi) = v(a_j(\pi))]$ denotes the fact that the value of the jth attribute along the path π is $v(a_j(\pi))$.

3.3.3 Horizontally Fragmented Distributed Data

When the data are horizontally distributed, examples corresponding to a particular value of a particular attribute are scattered across different locations. In order to identify the best split

at a node in a partially constructed tree, all the sites are visited and the counts corresponding to candidate splits of that node are accumulated. The learner uses these counts to find the attribute that yields the best split to further partition the set of examples at that node. Thus, given $L(\pi)$, in order to split the node corresponding to $a_l(\pi) = v(a_l(\pi))$, the statistics gathering component has to obtain the counts $t_k = count_{D_k|L(\pi)}(x)$, $t_j^k = count_{D_k|L(\pi)}(c_j)$, $t_{ij}^k = count_{D_k|L(\pi)}(a_i|c_j)$ for all $x \in D_k|L(\pi)$, for all classes c_j and all candidate attributes a_i that are not already in $L(\pi)$ (note that $D_k|L(\pi)$ is the subset of the data set D_k that satisfies the constraints specified by $L(\pi)$).

The algorithm for learning decision trees from horizontally fragmented data applies a refinement operator R_{DT} to refine a partial decision tree h_i based on the sufficient statistics collected from $D_1, \cdots, D_n$ given h_i. The refinement operator is applied $|T|$ times, where $|T|$ is the size of the decision tree (number of nodes). Initially the tree is null, but at each of the subsequent iterations h_i is the partial decision tree constructed so far (see Figure 3.8 for an example).

Most of the pseudocode in Figure 2.3 remains the same, except that we expand the procedure that determines the best attribute for a split, by showing how to do this when data are horizontally distributed. The pseudocode is shown in Figure 3.9.

Theorem 3.20. (Exactness) *The algorithm for learning Decision Trees from horizontally distributed data, shown in Figure 3.9, is* exact *with respect to its centralized counterpart, shown in Figure 2.3.*

Proof. As can be seen, the algorithm for learning from distributed data is similar to the algorithm for learning from centralized data, except the part where the best attribute is chosen from the candidate attributes. But the best attribute selection is based on counts and because the count operation is additive, computing the counts from distributed data and adding them up gives the same result as computing the counts when all data are together at a central place (Lemma 3.7). Thus, the distributed algorithm is exact. □

Theorem 3.21. (Time Complexity) *If both serial and parallel access to the data are allowed, then parallel access is preferred as it results in an algorithm for learning Decision Tree*

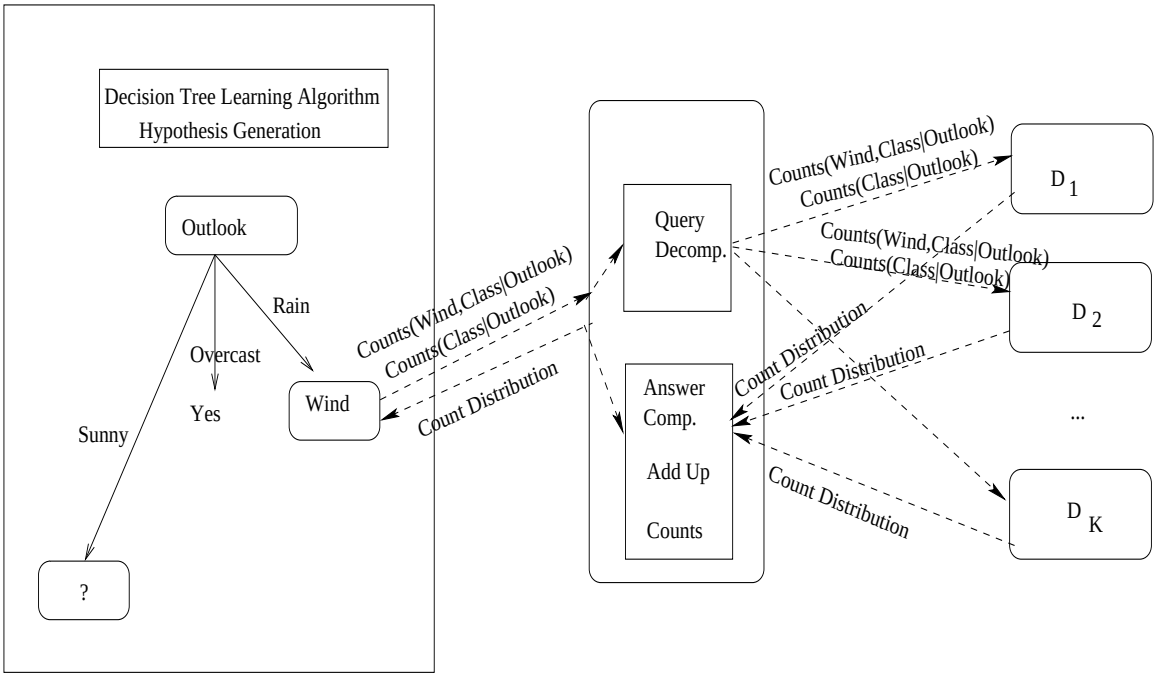

Figure 3.8 Learning Decision Tree classifiers from horizontally fragmented distributed data: for each node, the algorithm asks a joint count statistical query, the query is decomposed into sub-queries and sent to the distributed data sources, and the resulting counts are added up and sent back to the learning algorithm. One iteration is shown

Best Attribute from Horizontally Fragmented Data

BestAttribute$(D_1, \cdots, D_K, L(\pi))$
for (each data source D_k)
{
• Get $L(\pi)$ from the central location.
• Compute the counts $t^k = count_{D_k|L(\pi)}(x)$ of the examples $x \in D_k$ that satisfy the constraints specified by $L(\pi)$.
• Send these counts to the central location.
for (each class c_j)
{
• Compute the counts $t_j^k = count_{D_k|L(\pi)}(c_j)$ of the examples $x \in D_k$ that satisfy the constraints specified by $L(\pi)$.
• Send these counts to the central location.
for (each attribute value a_i that is not already used in $L(\pi)$)
{
• Compute counts $t_{ij}^k = count_{D_k|L(\pi)}(a_i|c_j)$ of the examples $x \in D_k$ that satisfy the constraints specified by $L(\pi)$.
• Send these counts to the central location.
}
}
}
At the central location:
for (each attribute that is not already used in $L(\pi)$)
{
• Compute $P(c_j) = \frac{\sum_{k=1}^K t_j^k}{\sum_{k=1}^K t^k}$, $P(a_i|c_j) = \frac{\sum_{k=1}^K t_{ij}^k}{\sum_{k=1}^K t_j^k}$.
• Compute information gain using the probabilities defined above.
}
Choose the attribute with the best information gain.

Figure 3.9 Decision Tree classifiers: finding the best attribute for split when data are horizontally fragmented

classifiers from horizontally distributed data, shown in Figure 3.9, that is K faster than the algorithm for learning Decision Tree classifiers from centralized data, shown in Figure 2.3

Proof. We can see that for both algorithms (Figure 3.9 and Figure 2.3) the computation of the counts for one node can be done with one pass through the data. However, in the distributed case, we can compute the counts at each location k independent of the counts at the other locations. Thus, if parallel data access is performed, the counts can be computed in parallel, which makes the algorithm for learning from distributed data K times faster then the algorithm for learning from centralized data. Therefore, parallel access to data is preferred to serial access to data. □

Theorem 3.22. (Communication Complexity) *Under the assumption that each data source allows shipping of raw data and computation of sufficient statistics, the algorithm for learning decision trees from horizontally distributed data, shown in Figure 3.9, (where we ship the statistics all the time) is preferable to the algorithm for learning from centralized data, shown in Figure 2.3 (where we bring all the data to the central location), in terms of communication complexity, if:* $O(|C||V||T|K) < O(|D|)$.

Proof. For each node in the decision tree T, the central location has to transmit the current path $L(\pi)$ (whose size is at most $2|A|$, from at most $|A|$ attribute-value pairs) to each site and each site has to transmit the counts $t^k = count_{D_k|L(\pi)}(x)$, $t^k_j = count_{D_k|L(\pi)}(c_j)$, $t^k_{ij} = count_{D_k|L(\pi)}(a_i|c_j)$ to the central location. The number of messages needed to ship these counts is at most $(1 + |C| + |C||A||V|)$. Hence, the total amount of information that is transmitted between sites is given by $(2|A|+1+|C|+|C||A||V|)|T|K$. The number of messages shipped in the centralized case when all the data are shipped is $(|D_1|+\cdots+|D_K|)(|A|+1) = |D|(|A|+1)$. Thus, the algorithm for learning from distributed data is preferred to the algorithm for learning from centralized data if $(2|A| + 1 + |C| + |C||A||V|)|T|K < |D|(|A| + 1)$, which implies $O(|C||V||T|K) < O(|D|)$. □

Observation 3.23. It is worth noting that the bounds presented in Theorem 3.22 can be further improved so that they depend on the height of the tree instead of the number of nodes in the

tree by taking advantage of the sort of techniques that are introduced in [Shafer *et al.*, 1996; Gehrke *et al.*, 1999] (we gather statistics once for each level of the tree as opposed to once for each node).

Observation 3.24. We notice that for a data source D_k we ship the statistics as opposed to data if $O(|C||V||T|) < O(|D_k|)$. This ensures that overall we get optimal performance, because the previous inequality implies $|C||V|T|K < |D_1| + \cdots |D_K| = |D|$.

Observation 3.25. We can see that the communication analysis performed above is a global analysis, which depends on the size of the final decision tree T. However, we do not know $|T|$ a priori, but we can bound it by $|V||A|$ as $|A|$ represents a bound for the height of the tree and $|V|$ represents a bound on the branching factor.

Observation 3.26. Theorem 3.19. assumes that in the distributed case, we ship the statistics all the time, as opposed to the centralized case, when we ship the whole data to the central location once. However, in practice it may be the case that in the beginning it is better to ship the statistics, but towards the end of the algorithm the size of the data that satisfies the constraints imposed by the partial decision tree built is smaller then the size of the sufficient statistics that still need to be shipped, in which case shipping data becomes preferable.

Thus, if we know the size of the data and the size of the statistics at each step, and besides we have a good estimate for the number of iterations left, we can optimize further the communication complexity. This optimization is possible because the amount of data or statistics shipped at each step is an upper bound on the amount of information that would need to be shipped at the next step. We can prove the following theorem:

Theorem 3.27. *For each data source k and each iteration l corresponding to the node to be expanded on a partial path $L(\pi) = [a_1(\pi) = v(a_1(\pi))] \wedge [a_2(\pi) = v(a_2(\pi))] \cdots [a_l(\pi) = v(a_l(\pi))]$, let $D_k^l \subset D_k$ be the subset of the data D_k satisfying the constraints imposed by $L(\pi)$, A^l be the set of attributes that are not already in $L(\pi)$ and let $|T^l|$ be an estimate of the number of iterations left until the end of the algorithm. If $O(|C||V||T^l|) < O(|D_k^l|/K)$ then it is preferable to ship the statistics at step l, otherwise it is preferable to ship the data.*

Proof. It follows from the communication complexity theorem and from the observation that the amount of information shipped at the current step is an upper bound on the amount of information shipped at the next step. □

3.3.3.1 Vertically Fragmented Distributed Data

When the data are vertically distributed, we assume that each example has a unique index associated with it. Sub-tuples of an example are distributed across different sites. However, correspondence between sub-tuples of a tuple can be established using the unique index. As before, given $L(\pi)$, in order to split the node corresponding to $a_l(\pi) = v(a_l(\pi))$, the statistics gathering component has to obtain the counts of examples that belong to each class for each possible value of each candidate attribute. Since each site has only a subset of the attributes, the set of indices corresponding to the examples that match the constraint $L(\pi)$ have to be made available at each site. To achieve that, first the central location sends the current path $L(\pi)$ to each site, then each site sends back to the central location the indices I_k of its data that satisfy the constraints. Second, the intersection $I = \cap_{k=1}^{K} I_k$ of these indices is found at the central location and sent back to the data sources. Also the number $t = count_I(x)$ of examples in D that satisfy the constraints is computed (this number is equal to $size(I)$). Using the set of indices I, each site can compute the relevant counts $t_{ij} = count_{D^k|I}(a_i|c_j)$ that correspond to the attributes that are stored at that site. One of the sites sends the counts $t_j = count_{D_k|I}(c_j)$ (they can be computed at any of the data sources). All these counts received from the sites are used to compute the information gain for the candidate attributes and thus to select the best attribute to further split the node corresponding to $a_l(\pi) = v(a_l(\pi))$.

Similar to the algorithm for learning decision trees from horizontally fragmented, the algorithm for learning from vertically distributed data applies a refinement operator R_{DT} to refine a partial decision tree h_i based on the sufficient statistics collected from $D_1, \cdots, D_n$ given h_i. The refinement operator is applied $|T|$ times, where $|T|$ is the size of the decision tree (number of nodes). Initially the tree is null, but at each of the subsequent iterations h_i is the partial decision tree constructed so far.

As in the horizontally distributed case, most of the pseudocode in Figure 2.3 remains the

same, except that we expand the procedure that determines the best attribute for a split, by showing how to do this when data are vertically distributed. The pseudocode that finds the best attribute in this case is shown in Figure 3.10.

Best Attribute from Vertically Fragmented Data

BestAttribute$(D_1, \cdots, D_K, L(\pi))$
Central location:
{
- Send current $L(\pi)$ to each data source D_k.
- Get back I_k from each data source D_k.
- Compute $I = \cap_{k=1}^{K} I_k$.
- Send I to each data source D_k.
- Compute $t = count_I(x)$.

}
for (each class c_j)
{
Compute $t_j = count_{D_k|I}(c_j)$ at any $k \in \{1, \cdots, K\}$.
Send t_j to the central location.
}
for (each data source D_k)
{
for (each class c_j)
for (each attribute value a_i at D_k that is not already used in $L(\pi)$)
- Compute counts $t_{ij}^k = count_{D_k|I}(a_i|c_j)$ of the examples $x \in D_k|I$.
- Send these counts to the central location.

}
At the central location:
for (each attribute that is not already used in $L(\pi)$)
{
- Compute $P(c_j) = \frac{t_j}{t}$, $P(a_i|c_j) = \frac{t_{ij}}{t_j}$.
- Compute information gain using the probabilities defined above.

}
Choose the attribute with the best information gain.

Figure 3.10 Decision Tree classifiers: finding the best attribute for split when data are vertically fragmented

Theorem 3.28. (Exactness) *The algorithm for learning Decision Trees from vertically distributed data, shown in Figure 3.10, is* exact *with respect to its centralized counterpart, shown in Figure 2.3.*

Proof. As can be seen, the distributed algorithm is similar to the centralized algorithm, except the part where the best attribute is chosen from the candidate attributes. But the best attribute selection is based on counts and because all the data about an attribute is at the same location, once we select the examples that satisfy the constraints in $L(\pi)$, the counts obtained are the same as those that would be computed if we brought all the data together. Thus, the algorithm for learning from vertically fragmented data is exact. □

Theorem 3.29. (Time Complexity) *If both serial and parallel access to the data are allowed, then parallel access is preferred as it results in an algorithm for learning Decision Tree classifiers from vertically distributed data, shown in Figure 3.10, that is K faster than the algorithm for learning Decision Tree classifiers from centralized data, shown in Figure 2.3*

Proof. We can see that for both algorithms (Figure 3.10 and Figure 2.3) the computation of the counts for one node can be done with one pass through the data. However, in the distributed case, we can compute the counts at each location k independent of the counts at the other locations. Thus, if parallel data access is performed, the counts can be computed in parallel, which makes the algorithm for learning from distributed data K times faster then the algorithm for learning from centralized data. Therefore, parallel access to data is preferred to serial access to data. □

Theorem 3.30. (Communication Complexity) *Under the assumption that each data source allows shipping of raw data and computation of sufficient statistics, the algorithm for learning decision trees from vertically distributed data, shown in Figure 3.9, (where we ship the statistics all the time) is preferable to the algorithm for learning from centralized data, shown in Figure 2.3 (where we bring all the data to the central location), in terms of communication complexity, if:* $O((|D| + |A||V||C|)K|T|) < O(|D|(|A| + K))$.

Proof. For each node in the decision tree T, the central location has to transmit the current path $L(\pi)$, whose size is at most $2|A|$ (because there are at most $|A|$ attribute-value pairs). Then the data sources send the set of indices of the examples that satisfy the constraints in $L(\pi)$ (the size of each set is at most $|D|$), and the central location sends back the set of indices

given by the intersection of the local indices sets (again, the size of this set is at most $|D|$). Finally one of the sites has to transmit $t_j = count_{D_k|L(\pi)}(c_j)$ (i.e., $|C|$ messages) and each site has to transmit $t_{ij} = count_{D_k|L(\pi)}(a_i|c_j)$ for the $|A_k|$ attributes at its location (i.e., $|V||A_k||C|$ messages). The total number of messages needed to be shipped in the distributed case is at most $(K|A| + K|D| + K|D| + |C| + |C||V||A|K)|T|$. The number of messages shipped in the centralized case when all the data are shipped is $|D|(|A_1|+2+\cdots+|A_K|+1) = |D|(|A|+K+1)$ (one site sends both the index and class value, the others send just the index value). Thus the algorithm for learning from distributed data is better than the algorithm for learning from centralized data if $(K|A|+K|D|+K|D|+|C|+|C||V||A|K)|T| < |D|(|A_1|+2+\cdots+|A_K|+1) = |D|(|A| + K + 1)$, which implies $O((|D| + |A||V||C|)K|T|) < O(|D|(|A| + K)$. □

Observation 3.31. It is worth noting that the bounds presented here can be further improved if we compute the statistics once for each level of the tree instead of each node by taking advantage of the sort of techniques that are introduced in [Shafer *et al.*, 1996; Gehrke *et al.*, 1999] (in which case they would depend on the height of the tree instead of the number of nodes in the tree).

Observation 3.32. We notice that for a data source D_k we ship statistics as opposed to data if $O((|D|/K + |A_k||V||C|)|T|) < O(|D||A_k|)$. This ensures that overall we get optimal performance because the inequality here implies $(|D| + |A||V||C|)T < |D||A|$ (we know $|A_1| + \cdots + |A_K| = |A|$).

Observation 3.33. We can see that the communication analysis performed here is a global analysis which depends on the size of the final decision tree T. However, as in the case of horizontally fragmented data, we don't know $|T|$ a priori, However, we can bound it by $|V||A|$ because $|A|$ represents a bound for the height of the tree and $|V|$ represents a bound on the branching factor.

Observation 3.34. The communication complexity theorem above assumes that in the distributed case, we ship the statistics at each step as opposed to the centralized case where we ship all data to the central location once. However, in practice it may be the case that in the beginning it is preferable to ship the statistics, but towards the end of the tree construction

the subset of the data that satisfies the constraints imposed by the partial decision tree built so far is smaller compared to the size of the sufficient statistics that need to be shipped until the end, in which case shipping the data becomes preferable.

Thus, if we know the size of the data and the size of the statistics that would need to be shipped at each step, and in addition, we have a good estimate for the number of iterations left, then we can optimize further the communication complexity. This optimization is possible because the amount of data or statistics at each step is an upper bound on the amount of information that would need to be shipped at the next step. We can prove the following theorem:

Theorem 3.35. *For each data source k and each iteration l corresponding to the node to be expanded on a partial path $L(\pi) = [a_1(\pi) = v(a_1(\pi))] \wedge [a_2(\pi) = v(a_2(\pi))] \cdots [a_l(\pi) = v(a_l(\pi))]$, let $D_k^l \subset D_k$ be the subset of the data D_k satisfying the constraints imposed by $L(\pi)$, A_k^l be the attributes at site k that are not already in $L(\pi)$, let $|T^l|$ be an estimate of the number of iterations left until the end of the construction of the tree T and $|D^l|$ be the size of the data (number of examples) shipped at iteration l. If $O((|D^l|/K + |A_k^l||V||C|)|T^l|) < O(|D^l||A_k^l|)$ then it is preferable to ship the statistics at step l, otherwise it is preferable to ship the data.*

Proof. It follows from the communication complexity theorem and from the observation that the amount of information shipped at each step is an upper bound on the amount of information shipped at the next step. □

3.3.4 Learning Threshold Functions from Distributed Data

We have seen in Chapter 2 that in the case of the perceptron algorithm, the value of the weight after one pass through the data can be seen as a minimal refinement sufficient statistic for the data set D, with respect to the partial hypothesis constructed in one iteration i of the algorithm. We denote by $\mathbf{w}_{i+1}(D)$ the value of the weight computed from D at iteration $i+1$. Then, $s(D, \mathbf{w}_i(D))$ is a refinement sufficient statistic for $\mathbf{w}_{i+1}(D)$. The output of the algorithm is the final weight $\mathbf{w}(D)$. In what follows, we will show how the sufficient statistics can be computed from horizontally and vertically distributed data.

3.3.4.1 Horizontally Fragmented Distributed Data

If we assume that the data D is horizontally distributed among the data sources $D_1, \cdots, D_K$, we can devise an algorithm for learning threshold functions from distributed data as in Figure 3.12. At each iteration i, the weight vector is subsequently sent to each data source D_k, updated based on the data at D_k, and then sent back to the central location together with a flag, which is true if no updates were made within one pass through that data (see Figure 3.11 for an example). The algorithm stops when all the flags are true after one complete pass through all the data sets.

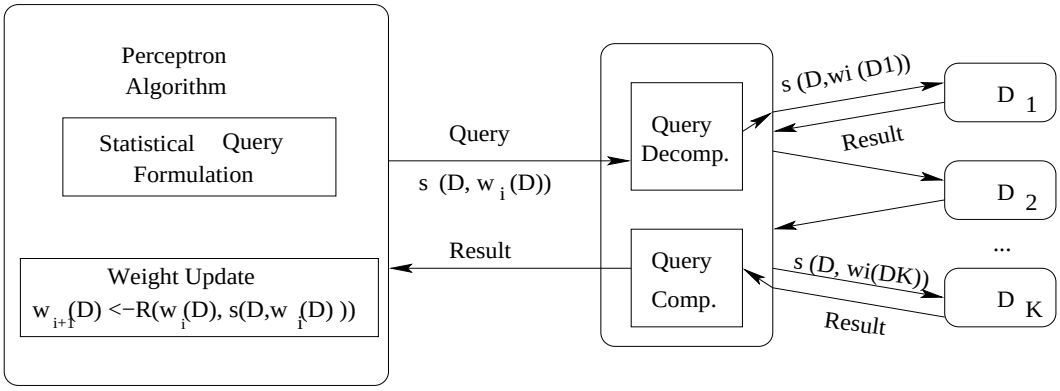

Figure 3.11 Learning Threshold Functions from horizontally distributed data: the algorithm asks a statistical query, the query is decomposed into sub-queries which are subsequently sent to the distributed data sources, and the final result is sent back to the learning algorithm. One iteration i is shown

Theorem 3.36. (Exactness) *The algorithm for learning threshold functions from horizontally distributed data, shown in Figure 3.12, is* exact *with respect to its centralized counterpart, shown in Figure 2.5.*

Proof. Same weight vector is computed in the distributed case. The only difference is that the weight is updated by visiting the data sources one by one, as opposed to the centralized case when all data are together and can be visited at once. □

Theorem 3.37. (Time Complexity) *The algorithm for learning threshold functions from horizontally distributed data, shown in Figure 3.12, has time complexity comparable to its centralized counterpart, shown in Figure 2.5.*

Learning Threshold Functions from Horizontally Distributed Data

Learning Phase

Initialize $\mathbf{w} \leftarrow [0, \cdots, 0]$ at the central location.
do
{
  for (all the data sets D_k, $i = 1, K$)
  {
    1. Set $flag_i \leftarrow false$.
    2. Send the current $\mathbf{w}$ to the data source D_k.
    3. Compute $\mathbf{w}(D_k)$ by updating $\mathbf{w}$ based on D_k.
    4. If (no change) $flag_i \rightarrow true$.
    5. Update $\mathbf{w} \leftarrow \mathbf{w}(D_k)$.
    6. Send the current value of $\mathbf{w}$ and the $flag_i$ back to the central location.
  }
}
until (a complete pass through data sets results in *no false flags*).
$\mathbf{w}^* \leftarrow \mathbf{w}$

Classification Phase

For a new instance $\mathbf{x}$

- assign $\mathbf{x}$ to the positive class if $\mathbf{x} \cdot \mathbf{w}^* > 0$;
- otherwise assign $\mathbf{x}$ to the negative class.

Figure 3.12 The Perceptron algorithm when data is horizontally fragmented

Proof. As opposed to algorithms for learning Naive Bayes classifiers or Decision Trees, for which the sufficient statistics at a local data source are independent of the sufficient statistics at the other data sources (and thus the sufficient statistics computation can be naturally parallelized), in the case of Perceptron algorithm the weight computation is a sequential process, which means that the weight vector cannot be updated at site $k+1$ before it was updated at site k. Hence, in this case the time complexity in the distributed case doesn't improve compared to the time complexity in the centralized case. □

Theorem 3.38. (Communication Complexity) *Under the assumption that at each iteration the data sources allow shipping raw data and also updating the weight vector locally, then the algorithm for learning threshold functions from distributed data, shown in Figure 3.12, is preferable to the algorithm for learning threshold functions from centralized data, shown in Figure 2.5, in terms of communication complexity, if* $O(MK) < O(|D|)$*, where* M *is the number of iterations executed by the algorithm.*

Proof. The size of an example $\mathbf{x}$ (and therefore the size of the weight vector $\mathbf{w}$) is $|A| = n$, and there are K data sets. The size of the data set D_k is $|D_k|$ and we know that $\sum_{k=1}^{K} |D_k| = |D|$. In the centralized case, we ship all the examples from the distributed sites to the central location once. Thus, the total amount of information shipped is $(|D_1|(|A|+1)+\cdots+|D_K|(|A|+1)) = (|A|+1)\cdot|D|$ If we assume that the algorithms stops after M iterations, then the amount of information shipped for each data source in the distributed case is $(2|A|+1)\cdot M$, because at each iteration the vector w is shipped from the central location to the local data source and then back from the local data source to the central location, together with the flag. Hence, the total amount of information shipped in the distributed case is $(2|A|+1)\cdot M\cdot K$. Thus, learning from distributed data is preferable to learning from centralized data if $(2|A|+1)MK < (|A|+1)|D|$, which implies $O(MK) < O(|D|)$. □

Observation 3.39. The amount of information shipped in the distributed case can be decreased by a factor of 2 if the initial weight is sent from the central site to the first data set D_1 and then the updated weight is sent from one data set D_k to the next data set D_{k+1} and so on. The process repeats til no weight updates are performed during one pass through all the data

sets (i.e., "serial" perceptron learning as opposed to "parallel" perceptron learning). When that happens, the current weight is sent back to the central location.

The communication complexity analysis performed above is a global analysis, which gives us the overall amount of information that needs to be shipped during the process of learning from distributed data. However, the whole process can be broken into iterations, as can be seen in the pseudocode in Figure 3.12. At each iteration either the weight vector is shipped, or the data are shipped for a data source.

We notice that for the perceptron algorithm we don't need to perform the optimization (i.e., decide if shipping data or statistics is preferable) at each iteration, but just once per data source. If we found out that shipping the weight is preferred to shipping the data (or the other way around) at the first step, this will remain true til the end of the algorithm, so we can reuse the results of the optimization for the first iteration.

However, if we do not take into account the maximum number of iterations when we perform the optimization at a certain iteration, it may turn out that although we shipped the smallest amount of information at each iteration, the overall amount of information shipped is not minimized. This is because $(2|A|+1) < |D_k|(|A|+1)$ does not imply $(2|A|+1) \cdot M < |D_k|(|A|+1)$ (we ship a new w at each iteration, for M iterations, but we ship data only once). We can prove the following theorem for the perceptron algorithm from distributed data:

Theorem 3.40. *If M is known or if it can be bounded, and for every data source k, we know the relationship between $(2|A|+1)M$ and $|D_k|(|A|+1)$ (i.e., either $(2|A|+1)M < |D_k|(|A|+1)$ or $(2|A|+1)M > |D_k|(|A|+1)$), then optimizing the amount of information shipped by each data source ensures that the amount of information shipped over all is optimal.*

Proof. We can assume without loss of generality that $(2|A|+1)M < |D_k|(|A|+1)$. Performing local optimization at each step means that at each iteration we ship w twice and the flag as opposed to shipping all the data D_k. Thus, we have $2|A|+1 < |D_k|(|A|+1)$. The amount of information shipped in one iteration for all the data sets is $(2|A|+1)K$, which means that the total amount of information shipped in M iterations is $(2|A|+1)KM$. But

$(2|A|+1)KM < (|A|+1)(|D_1|+\cdots|D_k|)$ (because $(2|A|+1)M < (|A|+1)|D_k|$), so we obtain global optimization. □

Observation 3.41. In practice, the number of iterations is unknown. However, we may be able to estimate (or bound) it based on the knowledge about the data domain, and thus check the inequality above, which tells us if learning from distributed data or learning from centralized data is preferable.

3.3.4.2 Vertically Fragmented Distributed Data

We assume that the data D is vertically distributed among the data sources $D_1, \cdots, D_K$. Given the incremental nature of the algorithm described in Figure 2.5, it is difficult to design an algorithm for learning threshold functions from distributed data whose communication complexity is better than the communication complexity of the algorithm for learning from centralized data. This is because we have to update the weight after seeing each example. To show how this could be done when data are vertically fragmented, we denote by $\mathbf{w}^k$ and $\mathbf{x}^k$ the projections of the current weight vector $\mathbf{w}$ and of the training example $\mathbf{x}$, respectively, on the attributes available at the data source D_k. At each iteration, for each example $\mathbf{x} \in D$ the central location sends $\mathbf{w}^k$ to each site k. The sites compute $< \mathbf{w}^k, \mathbf{x}^k >$ for $\mathbf{x}^k \in D_k$ and send these numbers back to the central location, where they are added up and the weight is updated accordingly. Because $|\mathbf{w}^1| + \cdots + |\mathbf{w}^K| = |\mathbf{w}| = |A| = |\mathbf{x}| = |\mathbf{x}^1| + \cdots + |\mathbf{x}^K|$, it is obvious that the amount of information shipped for one complete update of the weight vector (one iteration) is larger than the amount of information shipped in the centralized case. As the algorithm for learning from distributed data performs M iterations, it turns out that is very inefficient compared to the batch counterpart in terms of communication complexity. This proves that learning from distributed data, although always possible in principle, it it sometimes much worse than learning from centralized data.

However, there exists a batch variant of the Perceptron Algorithm [Nilsson, 1965] that can be easily transformed into an efficient algorithm for learning from vertically distributed data. In the Batch Perceptron, the weight update is done after one pass through all the

training examples. We can transform this algorithm into an algorithm for learning from vertically distributed data as follows: at each iteration the central location sends the current $\mathbf{w}^k$ to each site k. Each site k computes $\sum_{i=1}^{|D|} < \mathbf{w}^k, \mathbf{x}_i^k >$ and sends this number back to the central location, where all the numbers received from the distributed data sources are added up and the weight vector is updated. One of the sites has to send the class label to the central location as well. Thus, in this case the information shipped per iteration is $|\mathbf{w}^1|+1+\cdots+|\mathbf{w}^K|+1+|D| = |\mathbf{w}|+K+D = |A|+K+|D|$. Hence, the overall information shipped is $O((|A|+|D|+K)M)$ as opposed to $O(|A||D|)$. For practical problems, usually $(|A|+|D|+K)M < |A||D|$, so learning from distributed data is preferred to learning from centralized data. It is easy to see that the Batch Perceptron from vertically distributed data is exact with respect to its centralized counterpart because $\sum_{i=1}^{|D|} < \mathbf{w}, \mathbf{x}_i >= \sum_{i=1}^{|D|}\sum_{j=1}^{|A|} \mathbf{w}_j \mathbf{x}_{ij} = \sum_{i=1}^{|D|}\sum_{k=1}^{K} < \mathbf{w}^k, \mathbf{x}_i^k >$, so the same weight is obtained in both distributed and batch case. The algorithm for learning from distributed data is preferable in terms of time complexity because all the sums $\sum_{i=1}^{|D|} < \mathbf{w}^k, \mathbf{x}_i^k >$ can be computed in parallel.

3.3.5 Learning Support Vector Machines from Distributed Data

We have seen in Chapter 2 that the weight vector that defines the maximal margin hyperplane is a sufficient statistic for the SVM algorithm. Since this weight vector can be expressed as a weighted sum of a subset of training instances (called *support vectors*), the support vectors and the associated weights also constitute sufficient statistics for SVM. We will show how we can design efficient algorithm for learning SVM classifiers from distributed data under horizontal and vertical data fragmentation.

3.3.5.1 Horizontally Fragmented Distributed Data

In what follows, we will assume without loss of generality that the training examples are represented (if necessary, using a suitable kernel function) in a feature space in which the data $D = D_1 \cup \cdots \cup D_K$ is linearly separable.

A naive approach to distributed learning using SVM [Syed *et al.*, 1999] works as follows: apply the SVM algorithm for each data source D_k $(k = 1, K)$, and send the resulting support

vectors to the central site. At the central site, apply SVM algorithm to the union of the support vector sets received from the distributed data sources. The final set of support vectors and their corresponding weights are the sufficient statistics used to generate the separating hyperplane (see Figure 3.13 for an example). The pseudocode for this algorithm is shown in Figure 3.14.

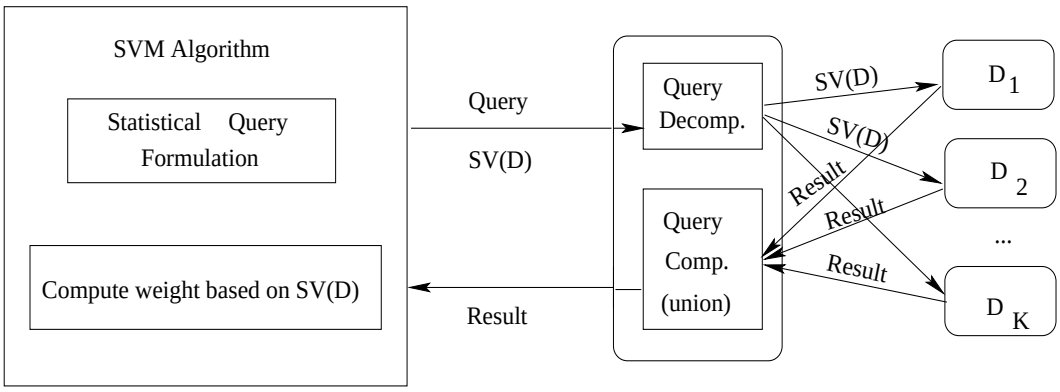

Figure 3.13 Learning SVM from horizontally distributed data: the algorithm asks a statistical query, the query is decomposed into sub-queries which are sent to the distributed data sources, the results are composed, and the final result is sent back to the learning algorithm

---

Naive SVM from Distributed Data
Learning Phase
for (each data source D_k)
{
Apply $SVM(D_k)$ and find the support vectors SV_k.
Send the support vectors SV_k to the central location.
}
At the central location:
Compute $D^{SV} = \cup_{k=1}^{K} SV_k$.
Apply $SVM(D^{SV})$
Let $< \mathbf{x}_{i_1}, y_{i_1} >, \cdots, < \mathbf{x}_{i_p}, y_{i_p} >$ be the set of final support vectors.
Let $\lambda_{i_l}^*$ be their corresponding weights.
Classification Phase
For a new instance $\mathbf{x}$
assign $\mathbf{x}$ to the class $f(\mathbf{x}) = \text{sign}(\sum_{l=1}^{p} y_{i_l} \lambda_{i_l}^* \cdot K(\mathbf{x}, \mathbf{x}_{i_l}) + b^*)$

Figure 3.14 Naive SVM from horizontally fragmented distributed data

Although this algorithm may work reasonably well in practice if the data sets $D_1, \cdots, D_K$ are individually representative of the entire training set D, in the sense that the hyperplane

determined by the support vectors derived from either one of them does not differ very much from that derived from the entire data set. However, if that is not the case, it can be shown that the resulting hyperplane can be an arbitrarily poor approximation of the target hypothesis [Caragea *et al.*, 2000]. This can be seen by considering the scenario illustrated in Figure 3.15. Here $D_1 = \{(-6,-2,+),(-2,-2,+),\ (6,-6,+),(-2,2,-),(2,2,-)\}$, and $D_2 = \{(-2,-2,+),(-2,-6,+),(2,2,-),(2,-2,-)\}$. Thus, the set $D_1 \cup D_2$ is clearly linearly separable. We can run the following experiment using an SVM algorithm (e.g., SVM^{light} [Joachims, 1999]:

- Apply SVM to $D_1 \cup D_2$ to get the support vector set
  $D_1 \cup D_2) = \{(-2,-2,+),(6,-6,+),(2,-2,-)\}$
- Apply SVM to D_1 to get the support vector set
  $SV_1 = \{(-6,-2,+),\ (-2,-2,+),(-2,2,-),(2,2,-)\}$
- Apply SVM to D_2 to get the support vector set
  $SV_2 = \{(-2,-2,+),(-2,-6,+),\ (2,2,-),(2,-2,-)\}$
- Apply SVM to $SV_1 \cup SV_2$ to get the support vector set
  $SV(SV_1 \cup SV_2) = \{(-2,-2,+),(-2,2,-),(2,-2,-)\}$

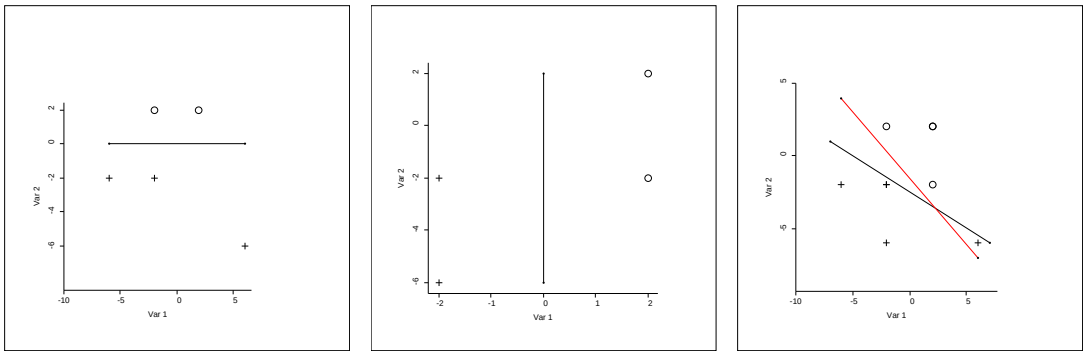

Figure 3.15 Counterexample to naive SVM from distributed data

Note that $SV(D_1 \cup D_2) \neq SV(SV_1 \cup SV_2)$. Because the separating hyperplane depends on the support vectors, this implies that the solution found by the SVM in the distributed setting is different from the solution found by batch learning. Thus, the naive approach to

learning SVMs from distributed data loses important boundary information (the data point $(6, -6, +)$ in the example above). As this depends on the underlying distribution over the pattern space, it can happen with an arbitrarily high probability, so the resulting classifier can have an arbitrarily high error. Therefore a better approach to designing learning algorithms that are effective in a distributed setting is necessary.

We would like to have $SV(D_1 \cup D_2) = SV(SV_1 \cup SV_2)$, but we have seen that the set of support vectors does not satisfy this property. However the convex hulls of the instances that belong to the two classes do satisfy this property [Gruber and Wills, 1993]. The convex hull of a set of points S, denoted $conv(S)$ is the smallest convex set containing S. That is,

$$conv(S) = \{\mathbf{X} \in R^N | \mathbf{X} = \sum_{\mathbf{X}_i \in S} \lambda_i \mathbf{X}_i, \sum \lambda_i = 1, \lambda_i \geq 0\}.$$

Thus, $conv(S)$ is the set of all non-negative affine combinations of points from S. If the set S is finite, the convex hull is a convex polyhedron given by the intersection of a finite number of closed half-spaces. We are interested in the vertices of this polyhedron because they uniquely define the convex hull.

Theorem 3.42. *Let $D_1, \cdots, D_K$ be convex sets. Then:*

$$conv(conv(D_1) \cup \cdots \cup conv(D_K)) = conv(D_1 \cup \cdots D_K).$$

Observation 3.43. Let $VConv(D)$ denote the vertices (training examples) that define the convex hull of a convex set D. It can be easily shown that:

$$VConv(VConv(D_1) \cup \cdots \cup VConv(D_K)) = VConv(D_1 \cup \cdots \cup D_K)$$

We assume that the data set $D = D_1 \cup \cdots \cup D_K$ is linearly separable (possibly through a kernel $K(.,.)$). Let $D_k(+)$ and $D_k(-)$ denote the positive and negative instances in the data set D_k. Similarly, let $D(+)$ and $D(-)$ denote the positive and negative instances in the data set D. Let $SVM(D)$ denote the result of applying the SVM algorithm to the data set D.

Similar to the algorithm in Figure 3.14, we can design an algorithm for learning SVMs from horizontally fragmented data using convex hulls as follows: each data source D_k computes $VConv(D_k(+))$ and $VConv(D_k(-))$ and sends these sets to the central location. At the

central location the SVM algorithm is applied to the union of all the sets of positive and negative convex hull vertices received from the distributed sites. The set of support vectors obtained and the weights associated with them represent the sufficient statistics that define the separating hyperplane. The pseudocode of this algorithm is shown in Figure 3.16.

SVM Learning From Horizontally Distributed Data
Learning Phase
for (each data source D_k)
{
Apply $VConv(D_k)$ and find the vertices on the convex hull of D_k.
Send the vertices $VConv(D_k)$ to the central location.
}
At the central location:
Compute $D^{conv} = \cup_{k=1}^{K} VConv(D_k)$
Apply $SVM(D^{conv})$
Let $< \mathbf{x}_{i_1}, y_{i_1} > \cdots , < \mathbf{x}_{i_p}, y_{i_p} >$ be the set of support vectors.
Let $\lambda^*_{i_l}$ be their corresponding weights.
Classification Phase
For a new instance $\mathbf{x}$
assign $\mathbf{x}$ to the class $f(\mathbf{x}) = \text{sign}(\sum_{l=1}^{p} y_{i_l} \lambda^*_{i_l} \cdot K(\mathbf{x}, \mathbf{x}_{i_l}) + b^*)$

Figure 3.16 Convex hull based SVM learning from horizontally fragmented distributed data

Theorem 3.44. (Exactness) *The algorithm for learning SVMs from vertically distributed data, shown in Figure 3.16, is* exact *with respect to its centralized counterpart, shown in Figure 2.8. (i.e., the set of support vectors found in the distributed setting is guaranteed to be identical to that obtained in the batch setting for any given training set).*

Proof. It follows immediately from Observation 3.43 and the fact that the set of support vectors is a subset of the convex hull vertices. □

Theorem 3.45. (Time Complexity) *The algorithm for learning SVMs from horizontally fragmented data, shown in Figure 3.16 is exponential in the number of dimensions n.*

Proof. The complexity of the convex hull computation has a linear dependence on the number of facets of the convex hull and the number of facets can be exponential in the dimension of the space [Gruber and Wills, 1993; Skiena, 1997]. This makes the algorithm in Figure 3.16 exponential in the number of dimensions n. Thus, this approach to learning SVMs from

horizontally distributed data is likely to be practical only when the convex hulls are simple (i.e., have relatively few facets) but not in general. □

Theorem 3.46. (Communication Complexity) *We assume that each data source allows both the shipment of the raw data and the computation of the convex hull at the local site. If M is the size of the largest subset of examples obtained as the result of the convex hull algorithm, then the algorithm for learning SVMs from horizontally distributed data, shown in Figure 3.16 is preferable to the algorithm for learning SVMs from centralized data, shown in Figure 2.8, in terms of communication complexity, if $O(KM) < O(|D|)$.*

Proof. In the distributed setting, each data source has to transmit at most M examples to the central site, hence the data shipped overall is $KM(|A| + 1)$. The data shipped in the centralized case is $|D|(|A| + 1)$. Thus, the algorithm for learning from horizontally distributed data is preferred to the algorithm for learning from centralized data if $O(KM) < O(|D|)$, which is usually the case in real world scenarios. □

We have described two algorithms for learning from horizontally distributed data, but one of them is not exact and the other one is not efficient. However, it is possible to design an efficient and exact algorithm for learning SVM-like classifiers from horizontally distributed data if we use a variant of the SVM algorithm, called Linear Support Vector Machines (LSVM) [Bradley and Mangasarian, 2000]. For simplicity, we describe this algorithm for linear separating surfaces (i.e., separating hyperplanes), but it can be extended to nonlinear separating surfaces the same way as the SVM algorithm is extended (i.e., by mapping the data to a higher dimensional space where a separating hyperplane can be found) [Bradley and Mangasarian, 2000].

We have seen that SVM selects from among the hyperplanes that correctly classify the training set, one that minimizes $\|\mathbf{w}\|^2$. This involves solving the following quadratic programming problem:

$\min_{\mathbf{w},b} \Phi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i = 1, \cdots, t.$

Linear Support Vector Machines algorithm [Bradley and Mangasarian, 2000] uses the ∞-norm

to measure the distance between the bounding planes, which leads to a linear programming formulation for the optimization problem, considerably less difficult than the quadratic optimization problem solved by SVM algorithm. *Support vectors* can be defined similar to the support vectors in SVM, and they are the only relevant data instances for computing the optimal separating plane.

We can transform LSVM from centralized data into an algorithm for learning from horizontally distributed data similar to the Linear Programming Chunking Algorithm (LPC) described in [Bradley and Mangasarian, 2000]. More precisely, this approach is similar to the naive approach to SVM from distributed data described above, except that several iterations through the distributed data sets are made. Thus, at each iteration, the central location sends the current set of support vectors to the distributed data sources. Each data source k adds those support vectors to its data and applies the LSVM algorithm to find a new set of support vectors given the global set of support vectors. The resulting set is sent back to the central location, which combines all the support vectors received from the distributed data sources and applies the LSVM algorithm to determine the new set of global support vectors. The pseudocode for this algorithm is shown in Figure 3.17.

We can directly apply the Theorem on the Finite Termination of LPC Algorithm in [Bradley and Mangasarian, 2000], to prove that this strategy yields a provably exact algorithm for learning an LSVM classifier from distributed data under horizontal fragmentation.

The communication complexity of this algorithm is similar to the communication complexity of the algorithm based on convex hull multiplied by the number of iterations. As in the case of perceptron, this is not known a priori, but we could bound it based on the prior knowledge about the domain.

Another way to design efficient and exact SVM-like classifiers is by considering algorithms which are theoretically proven to find the same solution as the SVM algorithm (e.g., Dual Perceptron Algorithm [Graepel and Herbrich, 2000], Logistic Regression [Zhang *et al.*, 2003a]), but lead to gradient descent optimization techniques, which can be efficiently performed in a distributed framework, similar to the classical Perceptron Algorithm.

LSVM from Horizontally Distributed Data
Learning Phase
Initialize $SV = \emptyset$ (the global set of support vectors).
repeat
{
Let $SV' = SV$.
Send SV' to all data sources D_k.
for (each data source D_k)
{
Apply $LSVM(D_k \cup SV')$ and find the support vectors SV_k.
Send the support vectors SV_k to the central location.
}
At the central location:
Compute $SV_D = \cup_{k=1}^{K} SV_k$.
Apply $LSVM(SV_D)$ to find the new SV

}
until $(SV = SV')$
Let $< \mathbf{x}_{i_1}, y_{i_1} >, \cdots, < \mathbf{x}_{i_p}, y_{i_p} >$ be the set of final support vectors.
Let $\lambda_{i_l}^*$ be their corresponding weights.
Classification Phase
For a new instance $\mathbf{x}$
assign $\mathbf{x}$ to the class $f(\mathbf{x}) = \text{sign}(\sum_{l=1}^{p} y_{i_l} \lambda_{i_l}^* \cdot K(\mathbf{x}, \mathbf{x}_{i_l}) + b^*)$

Figure 3.17 Exact and efficient LSVM learning from horizontally fragmented distributed data

3.3.5.2 Vertically Fragmented Distributed Data

Learning SVM classifiers from distributed data under vertical data fragmentation is more difficult than under horizontal data fragmentation. This difficulty arises from the fact that an entire instance (not just the projections on the available attributes) is needed at once in order to solve the optimization problem. It may be the case that it is not possible to design an algorithm for learning SVMs from vertically distributed data unless we look closely at the optimization problem that the SVM algorithm solves and ship enough information so that we can still solve it exactly. However, the amount of information transmitted in such a case is likely to be almost equal (or even greater than) to the size of data itself as in the case of incremental perceptron.

Fortunately, as we have seen that in the case of the Perceptron algorithm it is possible to learn efficiently from vertically distributed data if we consider the "batch" formulation of the algorithm. Similarly, Dual Perceptron Algorithm and Logistic Regression algorithms allow "batch" formulations that makes them more appropriate for a distributed framework than the SVM algorithm itself. As there are theoretical guarantees that prove that their solutions are comparable to the SVM solution, we thus obtain efficient algorithms for learning SVM-like classifiers from vertically distributed data.

3.3.6 Learning k Nearest Neighbor Classifiers from Distributed Data

In the case of k-NN algorithm, the learning phase consists simply of storing the data and the information extraction is done during the classification phase. As we have seen in Chapter 2, given a new example $\mathbf{x}$ to be classified, the distances to the closest k neighbors together with the labels of these neighbors represent minimal sufficient statistics for $h(\mathbf{x})$. As in the case of Naive Bayes, the sufficient statistics for k-NN classifiers can be computed in one step. In what follows, we will show how the minimal sufficient statistics can be computed when data are distributed, so that we obtain k-NN classifiers from distributed data.

3.3.6.1 Horizontally Fragmented Distributed Data

To compute the minimal sufficient statistics in the distributed setting, we compute the k nearest neighbors at each location l and ship the class labels corresponding to these neighbors (examples) together with the distance from each of them to the new example $\mathbf{x}$. Thus, we ship pairs $< d(\mathbf{x}, \mathbf{x}_{i_j}^l), c(\mathbf{x}_{i_j}^l) >$ for every nearest neighbor $\mathbf{x}_{i_j}^l$ at the location l (we denote by $d_{i_j}^l = d(\mathbf{x}, \mathbf{x}_{i_j}^l)$ the distance between x and $\mathbf{x}_{i_j}^l$ according to the metric used by the algorithm and by $c_{i_j}^l = c(\mathbf{x}_{i_j}^l)$ the class $y_{i_j}^l$ of the example $\mathbf{x}_{i_j}^l$. At the central location, we determine the k smallest distances among all the distances received and take a majority vote among the classes associated with those instances (see Figure 3.18 for an example). The majority class will be the class of the new example $\mathbf{x}$. The pseudocode for the distributed algorithm is shown in Figure 3.19.

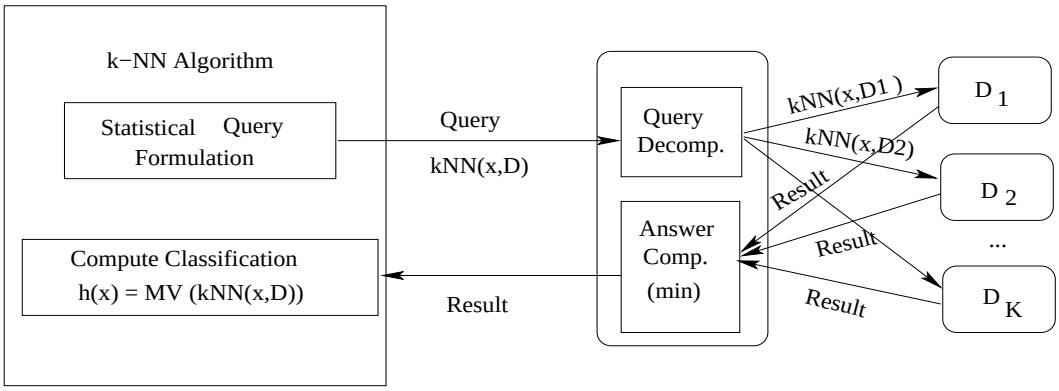

Figure 3.18 Learning k-NN classifiers from horizontally fragmented distributed data: the algorithm asks a statistical query, the query is decomposed into sub-queries which are sent to the distributed data sources, results are composed, and the final result is sent to the learning algorithm

Theorem 3.47. (Exactness) *The algorithm for learning k-NN classifiers from horizontally distributed data, shown in Figure 3.19, is* exact *with respect to the algorithm for learning k-NN classifiers from centralized data, shown in Figure 2.11.*

Proof. We need to show that the set of class labels used for majority vote in the distributed case is the same as the set of class labels used for majority vote in the centralized case. This follows from the observation that $min_k(X) = min_k(X_1 \cup \cdots \cup X_K) = min_k(min_k(X_1) \cup$

k-NN Classifiers from Horizontally Distributed Data

Learning Phase:

for (each data source D_l)
{
  for (each training example $(\mathbf{x}_i, y_i)$)
  {
    Add the example $(\mathbf{x}_i, y_i)$ to the list of training examples $TrEx(l)$ at location l.
  }
}

Classification Phase:

Given a new instance $\mathbf{x}$ to be classified:
  Send $\mathbf{x}$ to each site l.
  for (each data source D_l)
  {
    Let $\mathbf{x}^l_{i_1}, \cdots, \mathbf{x}^l_{i_k}$ be the k nearest neighbors of the instance x in the list $TrEx(l)$.
    Send $< d^l_{i_1}, c^l_{i_1} >, \cdots, < d^l_{i_k}, c^l_{i_k} >$ to the central location.
  }
  At the central location
    Compute the k nearest distances among all the distances received.
    Let $c_{i_1}, \cdots, c_{i_k}$ be the classes corresponding to these distances.
  return

$$h(\mathbf{x}) = \arg\max_{c \in C} \sum_{j=1}^{k} \delta(c, c_{i_j}),$$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise.

Figure 3.19 Algorithm for learning k Nearest Neighbors classifiers from horizontally fragmented distributed data

$\cdots \cup min_k(X_K))$, where $min_k(X)$ returns the k smallest distances in a set of distances X corresponding to the set of examples D and a new instance $\mathbf{x}$ to be classified. □

Theorem 3.48. (Time Complexity) *If both serial and parallel access to the data are allowed, then parallel access is preferred as it results in an algorithm for learning k-NN classifiers from horizontally distributed data, shown in Figure 3.19, that is K times faster than the algorithm for learning k-NN classifiers from centralized data, shown in Figure 2.11.*

Proof. We can see that for both algorithms (Figure 3.19 and Figure 2.11), the computation of the distances can be done with one pass through the data. However, in the distributed case, we can compute the the distances at each location k independent of the distances at the other locations. Thus, if parallel data access is performed, the distances can be computed in parallel, which makes the algorithm for learning k-NN classifiers from distributed data K times faster then the algorithm for learning from centralized data. Therefore, parallel access to data is preferred to serial access to data. □

Theorem 3.49. (Communication Complexity) *Under the assumption that the data sources allow shipping the raw data and also computation of the k smallest distances from a new example $\mathbf{x}$ to be classified to the training examples locally, the algorithm for learning k-NN classifiers from horizontally distributed data is preferable to the algorithm for learning k-NN classifiers from centralized data, in terms of communication complexity, if $O((|A| + k)K) < O(|A||D|)$.*

Proof. We compute the amount of information transmitted in the distributed case. First the central location sends the example $\mathbf{x}$ to be classified to all K distributed data sources. Each data source computes the k nearest neighbors and sends back their corresponding distances and classes. So the total amount of information transmitted is: $|A|K + 2kK = (|A| + 2k)K$. The total amount of information transmitted in the centralized case is $(|A| + 1)|D_1| + \cdots + (|A| + 1)|D_K| = (|A| + 1)|D|$. Thus, the algorithm from distributed data is preferred to the algorithm from centralized data if $(|A|+2k)K < (|A|+1)|D|$, which implies. $O((|A|+k)K) < O(|A||D|)$. □

Example 3.50. Assume a typical scenario: $|D| = 1,000,000$, $|A| = 100$, $k = 10$ and $K = 10$. Then the distributed k-NN sends 1200 messages as opposed to $100,000,000$ messages shipped in the distributed case.

3.3.6.2 Vertically Fragmented Distributed Data

In the vertically distributed data setting, a training example $(\mathbf{x}_i, y_i)$ is scattered over the distributed data sources. We denote by $\mathbf{x}^l$ the projection of an example $\mathbf{x}$ on the attributes at site l. To compute the distance $d(\mathbf{x}, \mathbf{x}_i)$ from a new example $\mathbf{x}$ to the training example $\mathbf{x}_i$, we need to compute $d_i^l = [d(\mathbf{x}^l, \mathbf{x}_i^l)]^2$ at each location l and ship them to the central location together with the index i and the class c_i. Then the distance $d(\mathbf{x}, \mathbf{x}_i) = \sqrt{\sum_{l=1}^{K} d_i^l}$ is computed at the central location for all the training examples $\mathbf{x}_i$ and the k smallest distances are found. We denote by $c_{i_1}, \cdots, c_{i_k}$ the classes corresponding to the k smallest distances. The class of the new example $\mathbf{x}$ is the majority class among these classes. The pseudocode for the algorithm for learning k-NN classifiers from vertically fragmented data is shown in Figure 3.20. We use the Euclidean distance to present the algorithm, but this can be generalized to any distance measure defined in Section 2.3.5.

Theorem 3.51. (Exactness) *The algorithm for learning k-NN classifiers from vertically distributed data, shown in Figure 3.20 is* exact *with respect to the algorithm for learning k-NN classifiers from centralized data, shown in Figure 2.11.*

Proof. We need to show that the set of class labels used for majority vote in the centralized case is the same as the set of class labels used for majority vote in the distributed case. It is obvious that the distances $d(\mathbf{x}, \mathbf{x}_i)$ computed in the distributed case are the same with the distances computed in the batch case. Then their corresponding sets of class labels are identical. □

Theorem 3.52. (Time Complexity) *If both serial and parallel access to the data are allowed, then parallel access is preferred as it results in an algorithm for learning k-NN classifiers from vertically distributed data, shown in Figure 3.20, that is K times faster than the algorithm for learning k-NN classifiers from centralized data, shown in Figure 2.11.*

k-NN Classifiers from Vertically Distributed Data

Learning Phase:

for (each data source D_l)
{
  for (each training example $(\mathbf{x}_i^l, y_i^l)$)
  {
    Add the example $(\mathbf{x}_i^l, y_i^l)$ to the list $TrEx(l)$.
  }
}

Classification Phase:

Given a new instance $\mathbf{x}$ to be classified:
  Send $\mathbf{x}$ to each site l.
  for (each data source D_l)
  {
  for (each example $\mathbf{x}_i^l$)
    {
      Let $d_i^l = [d(\mathbf{x}^l, \mathbf{x}_i^l)]^2$
      Send $< d_i^l, c_i^l, i >$ to the central location
    }
  }
At the central location
  for (each index i)
  {
    Compute $d_i = d(\mathbf{x}, \mathbf{x}_i) = \sqrt{\sum_{l=1}^{K} d_i^l}$
  }
  Let $c_{i_1}, \cdots, c_{i_k}$ be the classes corresponding to the k smallest distances.
  return

$$h(\mathbf{x}) = \arg\max_{c \in C} \sum_{j=1}^{k} \delta(c, c_{i_j}),$$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise.

Figure 3.20 Algorithm for k Nearest Neighbors classifiers from vertically fragmented distributed data

Proof. We can see that for both algorithms (Figure 3.20 and Figure 2.11) the computation of the distances can be done with one pass through the data. However, in the distributed case, we can compute distances at each location k independent of the distances at the other locations. Thus, if parallel data access is performed, the distances can be computed in parallel, which makes the algorithm for learning k-NN from vertically distributed data K times faster then the algorithm for learning from centralized data. Therefore, parallel access to data is preferred to serial access to data. □

Theorem 3.53. (Communication Complexity) *Under the assumption that the data sources allow shipping raw data and also computation of the k smallest distances from a new example* $\mathbf{x}$ *to be classified to the training examples, the algorithm for learning k-NN classifiers from vertically distributed data, shown in Figure 3.20, is preferable to the algorithm for learning k-NN classifiers from centralized data, shown in Figure 3.20, in terms of communication complexity, if* $O(|D|K) < O(|A||D|)$.

Proof. The amount of information transmitted in the distributed case is obtained as follows: the central location sends the projection $\mathbf{x}^l$ of the example $\mathbf{x}$ to be classified to all K distributed data sources. Each data source l computes d_i^l and sends back to the central location d_i^l, c_i^l, i for $i = 1, |D|$. Thus, the total amount of information transmitted is $(|A_1| + \cdots + |A_K|) + 3|D|K$. The total amount of information transmitted in the centralized case is $(|A| + 1)|D_1| + \cdots + (|A|+1)|D_K| = (|A|+1)|D|$. Thus, the algorithm for learning from vertically distributed data is better than the algorithm for learning from centralized data if $(|A| + 3|D|K < (|A| + 1)|D|$, which implies $O(|D|K) < O(|A||D|)$. □

Example 3.54. Assume a typical scenario: $|D| = 1,000,000$, $|A| = 100$, $k = 10$ and $K = 10$. Then the distributed k-NN sends $30,000,100$ messages as opposed to $100,000,000$ messages shipped in the distributed case. Although the communication complexity of the algorithm for learning k-NN from vertically distributed data is better than the communication complexity of the algorithm for learning from centralized case, the difference is not overwhelming as in the case of learning from horizontally fragmented data.

Observation 3.55. If $|A| \approx K$ then $O(|D|K) = O(|A||D|)$, hence learning from vertically distributed data and learning from centralized data are equally good in terms of communication. However, this is not usually the case, as the attributes are distributed at the K location. Obviously, $|A|$ cannot be larger than K in this scenario, so the algorithm k-NN classifiers from vertically distributed data should be always preferred to the algorithm for learning k-NN classifiers from centralized data.

3.4 Statistical Query Language

We have seen that learning from distributed data sources reduces to answering statistical queries from distributed data sources. Thus, it is important to define a statistical query language for formulating and manipulating statistical queries in a distributed setting. In this section, we define such a language consisting of a set of *operators* belonging to one of two categories: *data operators* and *statistical operators*, by extending the set of relational operators [Ramakrishanan and Gehrke, 2000].

Definition 3.56. *Data operators* correspond to operations whose inputs and outputs are sets of instances. They can be classified into:

- *Set Operators*: $\cup$, $\cap$, $-$, $\times$
- *Relational Operators*: SEL, $PROJ$, $JOIN$
- *Specialized Operators*: *HOR-INT*, *VER-INT*

Definition 3.57. *Statistical operators* are operators that output statistics about data. They can be classified into:

- *Aggregate Operators*: AVG, $COUNT$, $DIST$, MIN, MAX used to compute aggregate statistics for a data set.
- *Specialized Learning Operators*: SVM, DT, NN, $k-NN$ etc. used to extract algorithm-specific sufficient statistics from a data set.

- *Refinement/Combination Operators*: REF, REF^n, $COMB^n$ used to refine or combine current sufficient statistics.
- *Compositional Operators*: $+$, $UNION$, MIN, MAX, $VOTE$ etc. used to combine sufficient statistics extracted from several data sources.

3.4.1 Operator Definitions

Let $\mathcal{D}$ be the set of all possible data sets and $\mathcal{H}$ the space of the functions that a learner can draw on in order to construct classifiers. In a typical inductive learning scenario, $\mathcal{H}$ is a set of hypotheses. However, here it is useful to allow $\mathcal{H}$ to include not only the hypotheses but also sufficient statistics for hypotheses in $\mathcal{H}$.

Definition 3.58. Let $\mathcal{T} = \{\tau | \tau \text{ is a string}\}$ be a *set of types*. For each type τ, $dom(\tau) = \{v | v \text{ is a value of type } \tau\}$ is called the *domain* of τ.

We assume that every data source $D \in \mathcal{D}$ can be viewed as a *table* whose rows represent data *instances* and whose columns represent the *attributes* used to describe the instances.

Definition 3.59. Let $\{A_1, \cdots, A_n\}$ be the set of attributes used to describe the data in a particular data source D, and let $\{\tau_1, \cdots, \tau_n\}$ be the set of types associated with these attributes. The set $S = \{A_1 : \tau_1, \cdots, A_n : \tau_n\}$ is called the *schema* of the data source D.

Definition 3.60. We say that two schemas S_1 and S_2, corresponding to the data sets D_1 and D_2, respectively, are *union compatible* and we write $S_1 = S_2$ if they have the same number of attributes and the same types.

To define the query operators, we first define atomic conditions and selection conditions which are used by most of the operators to specify the precise data to which they are applied.

Definition 3.61. An *atomic condition* is defined as a condition having the form X op Y, where op $\in \{=, \neq, <, \leq, >, \geq\}$, and X, Y are terms (i.e., attributes or typed values $v : \tau$, with $v \in \text{dom}(\tau)$)).

Definition 3.62. A *selection condition* is a condition that can be defined recursively as follows:

- Any atomic condition is a selection condition.
- If c is a selection condition, then $\neg c$ is a selection condition.
- If c_1 and c_2 are selection conditions, then $c_1 \wedge c_2$, $c_1 \vee c_2$ are selection conditions.

3.4.1.1 Data Operators

Definition 3.63. The *union operator* is specified by $\cup : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$. It generates a new dataset D' from two existing dataset D_1 and D_2, where D' contains all the instances that occur either in D_1 or in D_2. To perform this operation, D_1 and D_2 should be union-compatible ($S_1 = S_2$) and the schema S' is identical to the common data source schemas. The union operator may perform the standard set union, multi-set union, or any suitably well-defined operation.

Definition 3.64. The *intersection operator* is specified by $\cap : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$. It generates a new dataset D' from two existing datasets D_1 and D_2, where D' contains all the instances that occur both in D_1 and in D_2. To perform this operation, D_1 and D_2 should be union-compatible ($S_1 = S_2$) and the schema S' is identical to the common data sources schemas.

Definition 3.65. The *set-difference operator* is specified by $- : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$. It generates a new dataset D' from two existing dataset D_1 and D_2, where D' contains all the instances that occur in D_1 but not in D_2. To perform this operation, D_1 and D_2 should be union-compatible ($S_1 = S_2$) and the schema S' is identical to the common data sources schemas.

Definition 3.66. The *cross product operator* is specified by $\times : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$. It generates a new data set D' from two existing datasets D_1 and D_2. The schema S' corresponding to the new data set D' contains all the fields in S_1 followed by all the fields in S_2, and the elements of D' are tuples $< e_1, e_2 >$, where $e_1 \in D_1$ and $e_2 \in D_2$.

Definition 3.67. The *selection operator* is specified by $\sigma : \mathcal{D} \rightarrow \mathcal{D}$. It generates a new dataset D' from an existing dataset D by selecting examples according to a specified criterion (e.g., it selects those instances from D that satisfy a selection condition on their attributes or it selects

instances according to a specified probability distribution). If S is the schema corresponding to the data set D, then D' will have the same schema S.

Definition 3.68. The *projection operator* is specified by $\pi : \mathcal{D} \rightarrow \mathcal{D}$. It generates a new dataset D' from an existing dataset D by projecting the examples in D on some attributes (it extracts columns from a table). If S is the schema corresponding to D, then the schema S' corresponding to D' has the property that $S' \subset S$.

Definition 3.69. The *join operator* is specified by $\bowtie$: $\mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$. It generates a new dataset D' from two existing data sets D_1 and D_2. The new data set D' is defined as $D' = D_1 \bowtie D_2 = \sigma_c(D_1 \times D_2)$, where c is a selection condition based on attributes from both S_1 and S_2.

Definition 3.70. A *horizontal integration operator* is specified by $\sqcup : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$. It generates a new data set D' from two existing data sets D_1 and D_2. It is defined similar to the union operator, except that a select and/or a project can be also performed on union result. Thus it is defined as $D' = D_1 \sqcup D_2 = \sigma_c(\pi_A(D_1 \cup D_2))$.

Definition 3.71. A *vertical integration operator* is specified by $\sqcap : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$. It generates a new data set D' from two existing data sets D_1 and D_2. It is defined similar to the join operator, except that here we assume that the two schemas S_1 and S_2 have at least one common attribute, and the join is performed on the common attribute. The schema S' contains the common attributes just once. Thus, V is defined as $\sqcap = \pi_{(S_1 \cup S_2)}(\sigma_{(A_i^1 = A_j^2)}(D_1 \times D_2))$, where $A_i^1 \in S_1$ and $A_j^2 \in S_2$ are pairs of common attributes.

3.4.1.2 Statistical Operators

Definition 3.72. An *aggregate operator* (average, count, distance, minimum, maximum etc.) is specified by $\psi : \mathcal{D} \rightarrow \mathcal{H}$, where the precise data set D' to which the operators are applied can obtained from the an existent data set D by applying selection and/or projection operators. The result could be a number, a sting, a matrix, etc. depending on the data to which it is applied.

Definition 3.73. A *learning operator* is specified by $L : \mathcal{D} \rightarrow \mathcal{H}$, where L denotes any inductive learning algorithm or information extraction algorithm. It takes as input a dataset

D and returns a function h that satisfies some specified criterion with respect to the data set. For example, if L is a consistent learner, it outputs a hypothesis that is consistent with the data. In other scenarios, L might compute relevant statistics from D.

Definition 3.74. The *refinement operator* $R_{D,L} : \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$, augments or refines a function h by incorporating a new statistics $s(D, h)$ according to the algorithm L. For instance, it may minimally modify a hypothesis according to L so as to be consistent with new data.

Definition 3.75. The *n-step refinement operator* $R^n_{D,L} : \mathcal{H}^n \times \mathcal{H} \rightarrow \mathcal{H}$, augments or refines a function h by incorporating a new statistics $s(D, h)$ according to the algorithm L in n steps.

Definition 3.76. The *combination operator* $C^n_{D,R} : \mathcal{H}^n \times \mathcal{H} \rightarrow \mathcal{H}$ produces a new function h by exploiting the information provided by the given functions $h_1, h_2, \cdots, h_n$ and a combination statistic $s(D, h_1, \cdots, h_n)$.

Definition 3.77. A *compositional operator* is specified by $\circ : \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$. It takes as input two elements of the same kind from $\mathcal{H}$ (statistics of data, e.g. two sets of examples satisfying some conditions, two count matrices, two numbers etc.) and it outputs a composition of these two elements (e.g., the union of the two sets or the addition of two matrices etc.).

Observation 3.78. This set of definitions is meant to be merely illustrative (and not exhaustive) with respect to the types of operators that might be useful in distributed settings. However, we will prove later that the set of operators introduced is complete with respect to a large class of algorithms that we are interested in.

Theorem 3.79. *The set of operators defined above is complete with respect to the learning algorithms considered in this dissertation, i.e., any statistical query needed by the learning algorithms of interest can be expressed using only operators defined above.*

Proof. By construction, it follows from the way we decomposed the algorithms of interest into information extraction and hypothesis generation components. □

3.5 Summary and Discussion

In this Chapter we have defined the problem of *learning from distributed data*, presented a general strategy for transforming algorithms for learning from data into algorithms for learning from distributed data and introduced criteria for comparing the two types of learning. This strategy is based on the decomposition of an algorithm into information extraction and hypothesis generation components. The information extraction from distributed data entails decomposing each statistical query q posed by the information extraction component of the learner into sub-queries $q_1, \cdots, q_K$ that can be answered by the individual data sources $D_1, \cdots, D_K$, respectively, and a procedure for combining the answers to the sub-queries into an answer for the original query q.

We have applied this strategy to design exact algorithms for learning Naive Bayes, Decision Trees, Threshold Functions, Support Vector Machines and k-NN classifiers from horizontally and vertically distributed data. We have compared the resulting algorithms with the traditional algorithms in terms of time and communication complexity.

Similar to the algorithms for learning Naive Bayes classifiers or Decision Trees, we can design algorithms for learning hypotheses from distributed data for a large class of algorithms that have counts as sufficient statistics: Bayesian Networks [Pearl, 2000; Jensen, 2001], Bags of Words [Mitchell, 1997], Relational Learning [Friedman *et al.*, 1999; Atramentov *et al.*, 2003], NB-k [Silvescu *et al.*, 2004a], Association Rules [Agrawal and Shafer, 1996] etc. Efficient ways for gathering count sufficient statistics are described in are described in [Graefe *et al.*, 1998; Gehrke *et al.*, 2000; Moore and Lee, 1998].

We have seen that in some cases, the algorithm for learning from centralized data is preferable to the algorithm for learning from vertically distributed data in terms of communication complexity (e.g. learning threshold functions). In other cases, the algorithm for learning from vertically distributed data is preferable to the algorithm for learning from centralized data (learning k-NN classifiers). In some other cases, the algorithms for learning from distributed data or the algorithms for learning from centralized data is preferable, depending on actual data parameters (size, dimension etc.) Sometimes the communication complexity depends

on the number of iterations, which is not known a priori (e.g., learning threshold functions). However, Krishnaswamy *et al.* [2002] propose methods for estimating the communication and time complexity a priori.

As can be seen in Section 1.4.1, there is a lot of related work on distributed learning. However, as opposed to the previous approaches, our approach can be applied to any learning algorithm and any kind of data fragmentation. Furthermore, as we usually ship sufficient statistics, some of the resulting algorithms can be easily applied in scenarios where privacy issues need to be taken into consideration.

Another major advantage of our approach is that it can be easily extended to an approach for learning from heterogeneous data, as we will see in the next Chapter.

4 LEARNING CLASSIFIERS FROM SEMANTICALLY HETERO-GENEOUS DATA

Our approach to learning classifiers from semantically heterogeneous distributed data is a natural extension of the approach to learning from distributed data discussed in Chapter 3, which assumes a common ontology shared by all of the data sources (See Figure 4.1 vs.Figure 3.3).

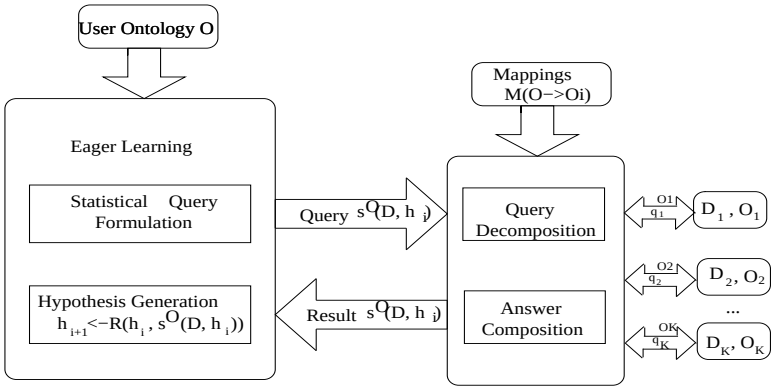

Figure 4.1 Learning from semantically heterogeneous distributed data: each data source has an associated ontology and the user provides a user ontology and mappings from the data source ontologies to the user ontology

In order to extend our approach to learning from distributed data sources into effective algorithms for learning classifiers from semantically heterogeneous distributed data sources, techniques need to be developed for answering statistical queries, posed by the learner in terms of the learner's ontology O, from the heterogeneous data sources (where each data source D_i has an associated ontology O_i). Thus, we have to solve a variant of the problem of integrated access to distributed data repositories, the data integration problem [Levy, 2000],

in order to be able to use machine learning approaches to acquire knowledge from semantically heterogeneous data.

4.1 Integration of the Data at the Semantic Level

As the number of data sources available for analysis increases every day, the need for tools for integrating these data sources becomes imperious. In the last few years, a lot of work in data integration community has focused successfully on data integration at the syntactic level. However, the integration at the semantic level is still an open problem.

In this section, we will describe a formal model, called *ontology-extended data sources* which allows us to do semantic data integration. Our model is inspired from a similar model called *ontology-extended relational algebra* described in [Bonatti *et al.*, 2003]. Although we can view a collection of physically distributed, autonomous, heterogeneous data sources as *though* they were relational databases [Reinoso-Castillo *et al.*, 2003], we will use the term *data sources* and not *relational databases* in what follows, to point out that, in principle, our data sources can be any kind of data sources (e.g., flat files, relational databases, web pages etc.), and therefore, the set of operations that can be executed at each data source is an extension of the set of operations allowed by relational databases.

Every data source that is used for learning has an implicit ontology associated with it. Intuitively, the ontology provides semantic information about the data source elements (e.g., attribute names, attribute values etc.) and about the relationships between these elements.

4.1.1 Motivating Example

We will consider the following example throughout this chapter.

Example 4.1. Suppose a company C_1 records information about weather in some region of interest R. From C_1's point of view, *Weather* is described by the attributes *Temperature*, *Wind*, *Humidity* and *Outlook*. An ontology O_1 associated with this data could tell us that *WindSpeed* is part of the *Wind* attribute description (called *part-of* relationship) and that *Sunny*, *Rainy*, *Cloudy* and *Snowy* are all *Outlook* descriptions (called *is-a* relationship). It

can also tell us that the *Temperature* is measured in *degrees Fahrenheit* and the *WindSpeed* is measured in *miles per hour*. The data D_1 that this company collects can be stored into a table like the one in Table 4.1.

Table 4.1 Data set D_1: Weather Data collected by company C_1

Day	*Temperature*	*WindSpeed*	*Humidity*	*Outlook*
1	20	16	67	*Cloudy*
2	10	34	53	*Sunny*
3	17	25	62	*Rainy*

Suppose that another company C_2 collects information about weather in the same region R. From C_2's point of view *Weather* is described by the attributes temperature denoted *Temp*, *Wind*, *Humidity* and precipitation denoted *Prec*. The ontology O_2 associated with its data tells us that *Speed* and *Direction* are both parts of the *Wind* attribute (*part-of* relationship) and that *Snow*, *Rain* and *NoPrec* are both *Prec* (*is-a* relationship). This ontology also stores information about the amount of precipitation by categorizing the precipitation values. For example, when recording the precipitation for one day, one can say *Rain* or *LightRain* or *HeavyRain*, etc. We say that *LightRain is-a* description of *Rain*. Furthermore, the ontology tells us that *Temp* is measured in *degrees Celsius* and that *WindSpeed* is measured in *kilometers per hour*. Thus, the data D_2 collected by this company looks like the one shown in the Table 4.2.

Table 4.2 Data set D_2: Weather Data collected by the company C_2

Day	*Temp*	*WindSp*	*WindDir*	*Humidity*	*Prec*
1	3	24	N	67	*Rain*
2	-2	50	NW	53	*LightRain*
3	0	34	NE	62	*NoPrec*

Suppose that a user U, having his or her own semantic about the weather domain, wants to infer some global information about weather in region R using the data collected by both C_1 and C_2. Assume that in this user ontology O_U, *Temperature* (measured in degrees Fahrenheit), *Wind* described by *WindSpeed* (measured in mph) and *WindDir*, *Humidity* and *Precipitation*

are the significant attributes. In order to be able to use simultaneously both data sources D_1 and D_2, the user needs to specify mappings from the data source ontologies O_1 and O_2 to the ontology O_U. For example, the user would map *Temperature* in O_1 and *Temp* in O_2 to *Temperature* in O_U ontology. The user needs also to specify a conversion function to convert *Temp* values in O_2 from degrees Celsius to Fahrenheit. Similarly, the user defines mappings and conversion functions for *WindSpeed*. With respect to *Precipitation*, the user observes that *Outlook* in O_1 and *Prec* in O_2 can be mapped to *Precipitation* in O_U. Also *Rainy* in O_1 can be mapped to *Rain* in O_U etc. We will see later what problems these mappings can generate and how we can deal with them.

A different user U' with a different semantic (ontology $O_{U'}$) may also want to use the data sources D_1 and D_2 for weather analysis. Similar to the first user, this user needs to specify mapping and conversion functions from the data source ontologies to his/her own ontology. Thus, every user can use the available data sources from his/her own perspective.

4.1.2 Ontology Definition

Having this example in mind, we will formally define the terms used above, by reformulating and extending the definitions in [Bonatti *et al.*, 2003] from relational databases to general data sources (represented as tables). We start with an informal definition for an ontology borrowed from philosophy:

Definition 4.2. "An *ontology* is an explicit formal specification of the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them."

Of particular interest are hierarchically structured ontologies [The Gene Ontology Consortium, 2000], [Bonatti *et al.*, 2003; Zhang and Honavar, 2003; Caragea *et al.*, 2004b]. Thus, to define ontologies formally, first we need to introduce the notion of a hierarchy:

Definition 4.3. Let S be a partially ordered set under ordering $\leq$. We say that another ordering $\preceq$ defines a *hierarchy* on S if the following three conditions are satisfied:

- (1) $x \preceq y \Rightarrow x \leq y,\ \forall x, y \in S$ (we say that $(S, \preceq)$ is *more concise than* $(S, \leq)$),

- (2) $(S, \leq)$ is the reflexive, transitive closure of $(S, \preceq)$,
- (3) no other ordering $\sqsubseteq$, which is more concise than $(S, \leq)$, satisfies (1) and (2).

Example 4.4. Let $S = \{Weather, Wind, WindSpeed\}$. We can define a partial ordering $\leq$ on S according to the *part-of* relationship. Thus, *Wind* is *part-of* the *Weather* characteristics, *WindSpeed* is also *part-of* the *Weather* characteristics, and *WindSpeed* is *part-of Wind* characteristics. In addition, everything is *part-of* itself. Therefore, $(S, \leq) = \{(Weather, Weather), (Wind, Wi$ $(WindSpeed, Weather), (WindSpeed, Wind)\}$. It follows that $(S, \preceq) = \{(Wind, Weather),$ $(WindSpeed, Wind)\}$, is the only one hierarchy associated with the order determined by the *part-of* relationship. Furthermore, $(S, \leq)$ is the reflexive, transitive closure of $(S, \preceq)$.

Let Λ be a finite set of strings that can be used to define hierarchies for a set of terms S. For example, Λ may contain strings like *is-a, part-of* corresponding to *is-a* and *part-of* relationships, respectively.

Definition 4.5. An *ontology* O (over terms in S) with respect to the partial orderings contained in Λ is a mapping Θ from Λ to hierarchies on S defined according to orderings in Λ.

In other words, an ontology associates orderings to their corresponding hierarchies. Thus, if *is-a*$\in \Lambda$, then Θ*(is-a)* will be the *is-a* hierarchy associated with the set of terms in S.

Example 4.6. Figures 4.2, 4.3 and 4.4 show the ontologies associated with the data sets D_1 and D_2, and the user ontology O_{U_1}, respectively, when $\Lambda = \{$*is-a, part-of*$\}$. In this case, the ontologies are *is-a* hierarchies H_1*(is-a)*, H_2*(is-a)*, H_U*(is-a)* and *part-of* hierarchies H_1*(part-of)*, H_2*(part-of)*, H_U*(part-of)*.

4.1.3 Ontology Integration

As mentioned before, we want to associate ontologies $O_1, \cdots, O_K$ with distributed data sources $D_1, \cdots, D_K$. For a user having an ontology O_U to be able to ask queries over several autonomous heterogeneous data sources, the user needs to specify mappings from the data

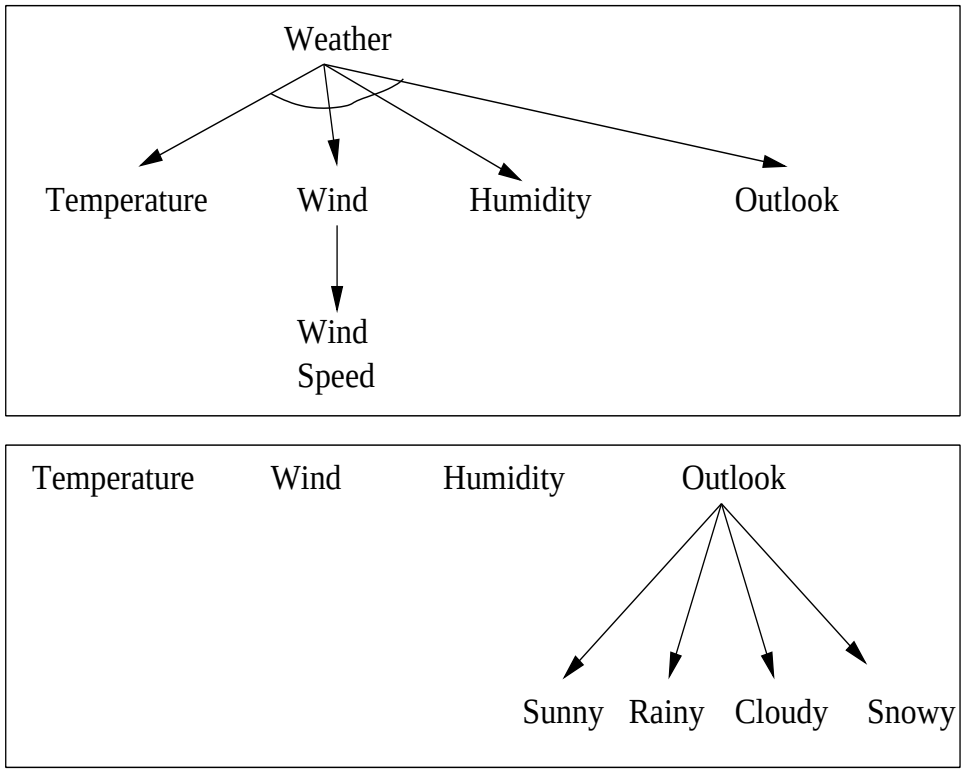

Figure 4.2 The ontology (*part-of* and *is-a* hierarchies) associated with the data set D_1

source ontologies $O_1, \cdots, O_K$ to the user ontology O_U, so that all the ontologies $O_1, \cdots, O_K$ are integrated according to the ontology O_U. We will formally define the integration ontology in what follows. To do that we start by explicitly associating types with all the objects that exist in an ontology.

In Chapter 3 we defined $\mathcal{T} = \{\tau | \tau \text{ is a string}\}$ to be a set of *types*. For each type τ, $dom(\tau) = \{v | v \text{ is a value of the type } \tau\}$ is called the *domain* of τ.

Observation 4.7. We can view all the internal nodes in a hierarchy as types whose domain is given by the values that their children can take. Some of these types are continuous types (e.g., *Temp*), others are enumerated types (e.g., *Outlook*).

Example 4.8. A type τ could be a predefined type, e.g., *int* or *string* or it can be a type like F^o (degrees Fahrenheit), USD (US Dollars), *mph* (Miles per hour) or it can be an enumerated type such as *Outlook* whose domain is given by the values: *Sunny, Rainy, Snowy* etc.

Definition 4.9. We denote by $\tau_{|d(\tau)}$ the restriction of the type τ to the sub-domain $d(\tau)$, where $d(\tau) \subseteq dom(\tau)$.

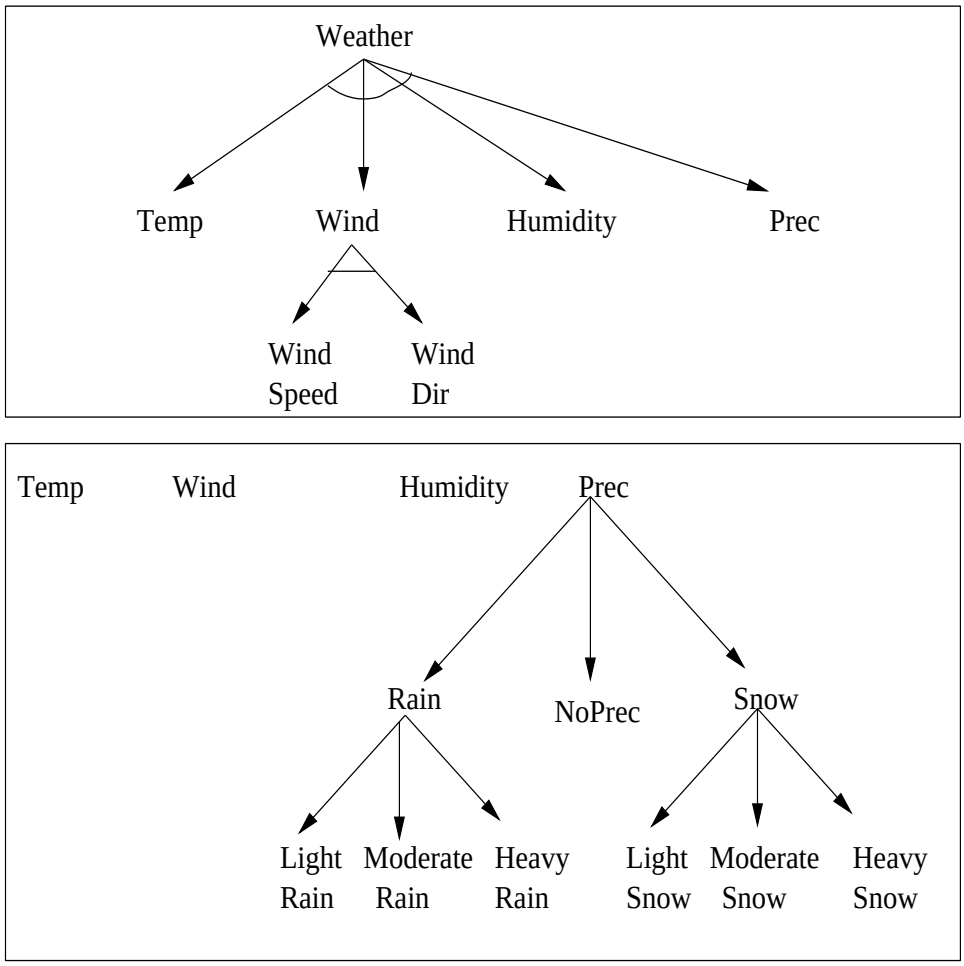

Figure 4.3 The ontology (*part-of* and *is-a* hierarchies) associated with the data set D_2

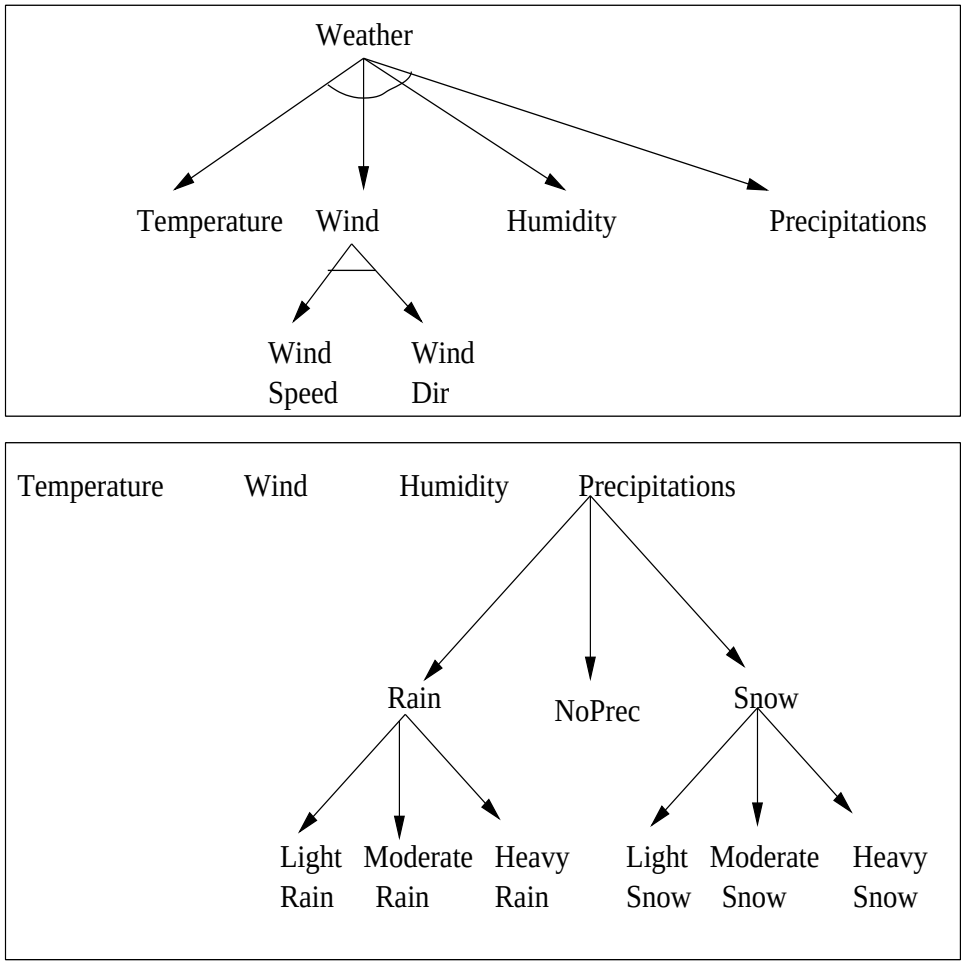

Figure 4.4 User ontology O_U, which represents an integration of the hierarchies corresponding to the data sources D_1 and D_2 in weather domain

Definition 4.10. Let $(H_1, \preceq_1), \cdots, (H_K, \preceq_K)$ be a set of K hierarchies determined by the same relationship *ord* (e.g., *is-a*) on the sets of terms $S_1, \cdots, S_K$, respectively, and let $(H_U, \preceq_U)$ be a user ontology determined by the relationship *ord* on a set of terms S. A *set of interoperation constraints IC(ord)* is a set of relationships that exist between elements from hierarchies H_i and elements from the hierarchy H_U. Thus, for two elements $x \in H_i$ and $y \in H_U$ we can have one of the following IC's: $x : H_i = y : H_U$ or $x : H_i \neq y : H_U$ or $x : H_i \leq y : H_U$ or $x : H_i \not\leq y : H_U$. If any or both x and y represent types in the corresponding hierarchy, we could also have constraints derived from the constraints above by restricting any or both types to a subdomain, i.e. $x_{|d(x)} : H_i = y : H_U$ or $x : H_i = y_{|d(y)} : H_U$ or $x_{|d(x)} : H_i = y_{|d(y)} : H_U$.

Example 4.11. For the weather example, if we consider the *is-a* hierarchies associated with the data sources D_1 and D_2 (i.e., H_1*(is-a)* and H_2*(is-a)*) and the *is-a* hierarchy H_U*(is-a)*, we have the following interoperation constraints, among others: $temp : H_2$*(is-a)* $= temperature : H_U$*(is-a)*, $humidity : H_1$*(is-a)*$\neq wind : H_U$*(is-a)*, $rainy : H_1$*(is-a)* $\not\leq lightRain : H_U$*(is-a)*, $heavyRain : H_2$*(is-a)*$\leq rain : H_U$*(is-a)*, $outlook_{|\{sunny,rainy,snowy\}} : H_1$*(is-a)*$= precipitations : H_U$*(is-a)* etc.

Definition 4.12. A *user perspective* UP with respect to a set of ontologies $O_1, \cdots, O_K$ is defined by a user ontology O_U and a set of interoperation constraints IC from hierarchies in $O_1, \cdots, O_K$ to hierarchies in user ontology O_U. We write $UP = (O_U, IC)$. In particular, the ontologies $O_1, \cdots, O_K$ and O_U could be simply hierarchies.

Definition 4.13. Let $(H_1, \preceq_1), \cdots, (H_K, \preceq_K)$ be a set of K hierarchies and $UP = (H_U, IC)$ a user perspective with respect to the hierarchies $H_1, \cdots, H_K$. We say that the hierarchies $H_1, \cdots, H_K$ are *integrable* according to the hierarchy $(H_U, \preceq)$ in the presence of the interoperation constraints IC (or equivalently H_U is the *integration hierarchy* of $H_1, \cdots, H_K$) if there exist K *injective partial mappings* $\phi_1, \cdots, \phi_K$ from $H_1, \cdots, H_K$, respectively, to H_U with the following two properties:

- For all $x, y \in H_i$, if $x \preceq_i y$ then $\phi_i(x) \preceq \phi_i(y)$ (we call this *order preservation*);
- For all $x \in H_i$ and $y \in H_U$, if $(x : H_i \text{ op } y : H_U) \in IC$, then $\phi_i(x)$ op y in the hierarchy H_U (we call this *interoperation constraints preservation*).

Definition 4.14. Let Λ be a set of strings (defining orderings) and $S_1, \cdots, S_K$ subsets of terms ordered according to the orderings in Λ; let $O_1, \cdots, O_K$ be ontologies with respect to Λ and $S_1, \cdots, S_K$, respectively, and $UP = (O_U, IC)$ a user perspective with respect to $O_1, \cdots, O_K$. We say that the ontologies $O_1, \cdots, O_K$ are *integrable* according to O_U (or equivalently, O_U is the *integration ontology* of $O_1, \cdots, O_K$) if and only if for each element $ord \in \Lambda$ the hierarchies $\Theta_1(ord), \cdots, \Theta_K(ord)$ are integrable according to $\Theta_U(ord)$.

Thus, a set of ontologies are integrable from a user perspective, if a set of mappings from the hierarchies in the local ontologies to the user hierarchies in the user ontology (satisfying the properties in the integration hierarchy definition) can be found.

Example 4.15. The ontologies O_1 and O_2 corresponding to the data sources D_1 and D_2 in the weather example (Figures 4.2 and 4.3) can be integrated according to the user ontology O_U (Figure 4.4).

We propose a simple algorithm for finding a set of mappings that witness the integration of the hierarchies $H_1, \cdots, H_K$ according to a user perspective $UP = (O_U, IC)$ (see Figure 4.5) and an algorithm for checking that the set of mappings found by this algorithm is consistent with the interoperation constraints and it satisfies the order preservation property (see Figure 4.6). We use these algorithms to integrate a set of ontologies $O_1, \cdots, O_K$ according to a user ontology O_U in the presence of the interoperation constraints $IC = \{IC(ord) | ord \in \Lambda\}$, by applying them to the set of hierarchies defined by each $ord \in \Lambda$ in the presence of $IC(ord)$.

Example 4.16. Let H_1, H_2 and H_U be the *is-a* hierarchies in Figures 4.2, 4.3 and 4.4, respectively. Let *IC(is-a)*= $\{Temp : H_2$*(is-a)* $= Temperature : H_U$*(is-a), Outlook* $: H_1$*(is-a)* $= Precipitation : H_U$*(is-a), Prec* $: H_2$*(is-a)*$= Precipitation : H_U$*(is-a), Sunny* $: H_1$*(is-a)*$= NoPrec : H_U$*(is-a), LightRain* $: H_2$*(is-a)*$\leq Rain : H_U$*(is-a),* $\cdots\}$. According to the first step of the Finding Mappings algorithm (name matching mappings), we add the mappings in Table 4.3. According to the second step of the algorithm (equality constraint mappings), we add the mappings in Table 4.4. By using Check Consistency algorithm, we can see that all the mappings constructed are consistent with the non-equality constraints and satisfy the order preservation property.

Finding Mappings
Input: a set of hierarchies $H_1, \cdots, H_K$ and a user perspective $UP = (H_U, IC)$.
Output: a mappings set MS.
{
$MS = \phi$
for (each H_i)
{

Name Matching Mappings:
for (each $term \in H_i$)
{
If ($term \in H_U$),then
$MS \rightarrow MS \cup \{term : H_i \rightarrow term : H_U\}$
(unless there is a constraint that does not allow this)
}

Equality Constraints Mappings:
for (each equality constraint $term_1 : H_i = term_2 : H_U$)
{
$MS \rightarrow MS \cup \{term_1 : H_i \rightarrow term_2 : H_U\}$
}
}
If (MS is consistent with the non-equality constraints)
return MS
Else
eliminate mappings that are inconsistent with the integrity constraints
return MS
}

Figure 4.5 Algorithm for finding mappings between a set of data source hierarchies and a user hierarchy

Check Consistency

Input: A set of mappings
$MS = \{term : H_i \rightarrow term' : H_U\}$ and a set of interoperation constraints
$IC = \{(term_1 : H_i \text{ op}_1 \ term'_1 : H_U), \cdots, (term_k : H_i \text{ op}_k \ term'_k : H_U)\}$.

Output: *true* if the MS is a set of partial injective mappings consistent with the set interoperation constraints and order preservation, *false* otherwise.

{

Check that MS is a set of mappings

for (each $term \in H_i$)
if ($(term : H_i \text{ op}' \ term' : H_U)$ & $(term : H_i \text{ op}'' \ term'' : H_U) \in IC$
& $(term' : H_U \neq term'' : H_U)$)
return *false*

Check that the mappings in MS are consistent

for (each $term \in H_U$)
if ($(term' : H_i \text{ op}' \ term : H_U)$ & $(term'' : H_i \text{ op}'' \ term'' : H_U) \in IC$
& $(term' : H_U \neq term'' : H_U)$)
return *false*

Check that the mappings in MS are consistent with the interoperation constraints IC

for (each $(term : H_i \rightarrow term' : H_U) \in MS$)
for (each $(term_k : H_i \text{ op}_k \ term'_k : H_U) \in IC$)
if $(term == term_k)$ & $(term' == term'_k)$
if (op_k is $\neq$)
return *false*

Check that the mappings in MS are consistent with the order preservation constraints IC

for (each t term such that $t : H_i \leq term : H_i$ or $term : H_i \leq t : H_i$)
if ($t : H_i$ maps to $T : H_U$ and $t : H_i \leq term : H_i$)
if $(T : H_U \geq term' : H_U)$ **return** *false*
else $(t : H_i \leq term : H_i)$
if $(term' : H_U \geq T : H_U)$ **return** *false*

return *true*

Figure 4.6 Algorithm for checking the consistency of a set of partial injective mappings with a set of an interoperation constraints and with the order preservation property

Table 4.3 Mappings from H_1*(is-a)* and H_2*(is-a)* (corresponding to the data sets D_1 and D_2) to H_U*(is-a)* found using name matching strategy

ϕ_1	ϕ_2
$Temperature \rightarrow Temperature$	-
$Wind \rightarrow Wind$	$Wind \rightarrow Wind$
$Humidity \rightarrow Humidity$	$Humidity \rightarrow Humidity$
-	$Rain \rightarrow Rain$
-	$LightRain \rightarrow LightRain$
-	$ModerateRain \rightarrow ModerateRain$
-	$HeavyRain \rightarrow HeavyRain$
-	$LightSnow \rightarrow LightSnow$
-	$ModerateSnow \rightarrow ModerateSnow$
-	$HeavySnow \rightarrow HeavySnow$
-	$NoPrec \rightarrow NoPrec$

Table 4.4 Mappings from H_1*(is-a)* and H_2*(is-a)* (corresponding to the data sets D_1 and D_2, respectively) to H_U*(is-a)* found from equality constraints

ϕ_1	ϕ_2
-	$Temp \rightarrow Temperature$
$Outlook \rightarrow Precipitation$	$Prec \rightarrow Precipitation$
$Sunny \rightarrow NoPrec$	-
$Rainy \rightarrow Rain$	-

Once a set of mappings is found using the algorithms described in Figures 4.5 and 4.6, the user is given the opportunity to inspect the mappings and add other mappings if needed and if they do not violate the interoperation constraints or the order preservation property.

4.1.4 Ontology-Extended Data Sources

So far, we have defined ontologies, explained what it means to integrate ontologies and showed how a user can check if his or her ontology can be an integration for a set of ontologies associated with autonomous data sources. Once the user integration ontology is defined (together with the mapping to the data sources ontologies), the user's goal is to ask queries in his/her ontology and get sound and complete answers from the data sources. For example, in the weather example, the user may want to ask queries about the days when the *Temperature* was higher than 40F. To get the answer to such a query, besides name mappings ($Temp : O_2 \rightarrow Temperature : O$), a conversion from degree Celsius to degree Fahrenheit is needed in the case of the second data source D_2.

In what follows, we will show how the information about ontologies can be incorporated into the associated data sources and also into the operations allowed by these data sources, so that we ensure that the answers to queries posed by a user are sound and complete.

Definition 4.17. We say that a total function $\tau_1 2 \tau_2 : \text{dom}(\tau_1) \rightarrow \text{dom}(\tau_2)$ that maps values of τ_1 to values of τ_2 is a *conversion function* from τ_1 to τ_2. The set of all conversion functions must satisfy the following constraints:

- For every two types τ_i, $\tau_j \in \mathcal{T}$ there exists at most one conversion function $\tau_i 2 \tau_j$.
- For every type $\tau \in \mathcal{T}$, $\tau 2 \tau$ exists (the identity function).
- If $\tau_i 2 \tau_j$ and $\tau_j 2 \tau_k$ exist, then $\tau_i 2 \tau_k$ exists and $\tau_i 2 \tau_k = \tau_i 2 \tau_j \circ \tau_j 2 \tau_k$.

Definition 4.18. We say that τ_1 can be *converted* into τ_2 and we write $\tau_1 \rightarrow \tau_2$ if there exists a conversion function $\tau_1 2 \tau_2$.

Observation 4.19. If τ_1 and τ_2 are on the same path in a hierarchy $(H, \leq)$ and $\tau_1 \leq \tau_2$, then $\tau_1 \rightarrow \tau_2$, which means that $\tau_1 2 \tau_2$ exists. (This is usually the identity.)

A user needs to specify conversion functions for all the ontology mappings defined in the system. If a conversion function is not explicitly specified, it is assumed to be the identity function.

Example 4.20. The conversion function associated with the mapping $Humidity : O_1 \rightarrow Humidity : O_U$ is the identity.

Example 4.21. The conversion function associated with the mapping $Temp : O_2 \rightarrow Temperature : O_U$ (where *Temp* is measured in degrees Celsius and *Temperature* is measured in degrees Fahrenheit) is the function $Temp(C)2Temperature(F)$ which converts Celsius to Fahrenheit.

Example 4.22. The conversion function associated with the mapping $Outlook : O_1 \rightarrow Precipitation : O_U$ (where $dom(Outlook) = \{Sunny, Rainy, Cloudy, Snowy\}$ and $dom(Precipitation) = \{Rain, Snow, NoPrec\}$) is a function $Outlook2Precipitation$ which converts values in $dom(Outlook)$ to values in $dom(Precipitation)$. Thus, $Outlook2Precipitation$ could "convert" *Sunny* to *NoPrec*, *Rainy* and *Cloudy* to *Rain*, and *Snowy* to *Snow*.

Observation 4.23. If the interoperation constraints are defined on subdomains of some domains, then the conversion functions are defined with respect to the respective subdomains.

Definition 4.24. Let H be a hierarchy and τ a type in that hierarchy. We denote by $\text{below}_H(\tau)$ the union between the values of τ and the subtypes τ' of τ:

$$\text{below}_H(\tau) := \{\tau' | \tau' \in H, \tau' \leq_H \tau\} \cup \text{dom}(\tau).$$

Example 4.25. We have: $\text{below}_H(Prec) = \{Rain, NoPrec, Snow, LightRain, ModerateRain, HeavyRain, LightSnow, ModerateSnow, HeavySnow\}$.

Definition 4.26. Let τ_1 and τ_2 be two types. A type τ is called the *least common supertype* of τ_1 and τ_2 if

- $\tau_1 \rightarrow \tau$ and $\tau_2 \rightarrow \tau$.

- If there exists τ' such that $\tau_1 \rightarrow \tau'$ and $\tau_2 \rightarrow \tau'$, then $\tau \rightarrow \tau'$.

Example 4.27. Let $X = Rain$ and $Y = HeavySnow$ be two terms in the *is-a* hierarchy of the user ontology in the *Weather* example. Then the least common supertype of $type(X)$ and $type(Y)$ is $Precipitation$.

We view any data source as a *table* whose lines represent data *instances* and whose columns represent the *attributes* used to describe the instances. Let $\{A_1, \cdots, A_n\}$ be the set of attributes used to describe the data in a particular data source D, and let $\{\tau_1, \cdots, \tau_n\}$ be the set of types associated with these attributes. The set $\{A_1 : \tau_1, \cdots, A_n : \tau_n\}$ is called the *schema* of the data source D.

Definition 4.28. Two schemas $S_1 = (A_1 : \tau_1^1, \cdots, A_n : \tau_n^1)$ and $S_2 = (A_1 : \tau_1^2, \cdots, A_n : \tau_n^2)$ are *compatible* if τ_i^1 and τ_i^2 have a *least common supertype* τ_i and the conversion functions $\tau_i^1 2 \tau_i$ and $\tau_i^2 2 \tau_i$ exist for all $i = 1, \cdots, n$. The common schema $S = (A_1 : \tau_1, \cdots, A_n : \tau_n)$ is called the *least common super-schema* of S_1 and S_2. The conversion functions $S_j 2 S$ are defined by $S_j 2 S(D) = \{(\tau_1^j 2 \tau_1(x_1), \cdots, \tau_n^j 2 \tau_n(x_n)) | (x_1, \cdots, x_n) \in D\}$ for $j = 1, 2$.

We will show that we can ensure the semantical correctness of an answer to a query if we extend each data source with its corresponding ontology and also with the type information associated with each attribute (i.e., data source schema), and specify conversion functions between different types.

Definition 4.29. We say that (D, S, O) is an *ontology-extended data source* if D is a data source (represented as a table), O is an ontology over D, $S = \{A_1 : \tau_1, \cdots, A_n : \tau_n\}$ is the data source schema, and the following conditions are satisfied:

(1) $\tau_1, \cdots, \tau_n \in O$ are types in the ontology O and

(2) $D \subseteq \text{below}_O(\tau_1) \times \cdots \times \text{below}_O(\tau_n)$.

Definition 4.30. Let $\mathcal{D} = \{D, (A_1 : \tau_1, \cdots, A_n : \tau_n), O\}$ be an extended data source and let X be a term (attribute, type or typed value) in the context of $\mathcal{D}$. We define the *type of the term* X as follows: $type(X) = \begin{cases} \tau_i & \text{if} \quad X = A_i \\ \tau & \text{if} \quad X = \tau \text{ or } X = v : \tau \end{cases}$

Example 4.31. Let $\mathcal{D}_1 = \{D_1, (Day : Day, Temp : Temp(F), WindSpeed : WindSpeed(mph), Humidity : PosInt, Outlook : Outlook), O_1\}$. Then, if $X = Temp$, we have $type(X) = Temp(F)$; if $X = Day$, then $type(X) = Day$; if $X = 16 : mph$, then $type(X) = WindSpeed(mph)$; if $X = Humidity$, then $type(X) = PosInt$ etc.

So far, we have extended data sources with ontologies and type information. We want to use these extended data sources to answer statistical queries, which means we need to show how to extend the operators defined in Section 3.4, so that we guarantee that the answers that we get to queries are sound and complete. To do that, we first re-define selection conditions which are used to specify the precise data to which the operators are applied. Thus, given the type information, we extend the definition of an atomic condition introduced in Chapter 3 as follows:

Definition 4.32. An *atomic condition* is defined as a condition having the form X op Y, where op $\in \{=, \neq, <, \leq, >, \geq,$ is-a, part-of, instance-of, subtype-of, above-type, below-type$\}$, and X, Y are terms (i.e., attributes, types or typed values $v : \tau$, with $v \in \text{dom}(\tau)$)).

Definition 4.33. A *selection condition* is a condition that can be defined recursively as follows:

- Any atomic condition is a selection condition.
- If c is a selection condition, then $\neg c$ is a selection condition.
- If c_1 and c_2 are selection conditions, then $c_1 \wedge c_2$, $c_1 \vee c_2$ are selection conditions.

Example 4.34. The following expressions are selection conditions: *type* part-of *Wind*, *Temperature>10:*
type part of *Wind* $\wedge$ *Temperature<10:F*.

Definition 4.35. An atomic condition X op Y, where op $\in \{=, \neq, <, \leq, >, \geq\}$ is *well-typed* if X and Y have a least common supertype τ and the conversion functions $type(X)2\tau$ and $type(Y)2\tau$ exist. If op $\in$ {is-a, part-of, instance-of, subtype-of, above-type, below-type}, then an atomic condition X op Y is always well-typed. A selection condition is well-typed if all its atomic conditions are well-typed.

Definition 4.36. We define the *value* of a term X with respect to an instance $t \in D$ as follows:

$$val_t(X) = \begin{cases} val_t(A_i) & \text{if} \quad X = A_i \\ \tau & \text{if} \quad X = \tau \notin \{A_1, \cdots, A_n\} \\ v & \text{if} \quad X = v : \tau \end{cases}$$

Example 4.37. Let $\mathcal{D}_1 = \{D_1, (Day : Day, Temp : Temp(F), WindSpeed : WindSpeed(mph), Humidity : PosInt, Outlook : Outlook), O_1\}$ be an extended data set and $t = \{3, 52, 30, 67, Sunny\}$ an instance in D_1. If $X = Outlook$, then $val_t(X) = Sunny$; if $X = temp$, then $val_t(X) = 52$; if $X = Integer$, then $val_t(X) = Integer$; if $X = 57 : Temp(F)$, then $val_t(X) = 57$.

Definition 4.38. An instance $t \in D$ *satisfies a well-typed condition* c in the context of $\mathcal{D}$ and we write $\mathcal{D}, t \models c$ if one of the following conditions is satisfied:

- $c = X$ op Y, where op $\in \{=, \neq, <, \leq, >, \geq\}$, and there exists τ the least common supertype of X and Y such that $(type(X)2\tau)(val_t(X))$ op $(type(Y)2\tau)(val_t(Y))$ is true.
- $c = X$ instance-of Y, $val_t(Y) \in \mathcal{T}$, $type(X) \rightarrow_O val_t(Y)$ and $val_t(X) \in dom(val_t(Y))$.
- $c = X$ subtype-of Y, $val_t(X) \in \mathcal{T}$, $val_t(Y) \in \mathcal{T}$, $val_t(X) \rightarrow_O val_t(Y)$.
- $c = c_1 \wedge c_2$, $\mathcal{D}, t \models c_1$ and $\mathcal{D}, t \models c_2$.
- $c = c_1 \vee c_2$, and either $\mathcal{D}, t \models c_1$ or $\mathcal{D}, t \models c_2$.
- $c = \neg c_1$ and $\mathcal{D}, t \not\models c_1$.
- $c = X$ below-type Y, $\mathcal{D}, t \models X$ instance-of $Y \wedge X$ subtype-of Y.
- $c = X$ above-type Y and $\mathcal{D}, t \models Y$ below-type X.

4.2 Ontology-Extended Query Operators

Definition 4.39. Let $(D_1, S_1, O_1), \cdots, (D_K, S_K, O_K)$ be a set of ontology-extended data sources and O an integration ontology for $O_1, \cdots, O_K$. An *ontology-extended data set X in the integrated domain* (o.e.d.s) is a data set (extended with its schema and the associated

ontology) whose instances are obtained from the set of instances of other data set in the domain by applying compositional operators and taking into account ontological information. Thus, we need to make sure that ontology-extended data sets are well-typed.

Definition 4.40. A *statistic over an ontology-extended data set* X (s.o.e.d.s.) is defined as the result of applying any function (operator or composition of operators) to that particular data set X. For example, a statistic can be a set of instances (i.e., another ontology-extended data set), counts of the instances in the initial data set, the average of a set of instances etc. For a statistic to be well-typed, we need to ensure that the ontology-extended data set that we start with are well-typed, and also that the result of applying the operators is well-typed.

In what follows we will show how to define well-typed ontology-extended data set and statistics recursively, i.e., we will show how to use the query operators in the context of extended data sources (types and ontological information) in a distributed environment.

4.2.1 Ontology-Extended Primitive Operators

Let $(D_1, S_1, O_1), \cdots, (D_K, S_K, O_K)$ be K ontology-extended data sources, and let O be an integration ontology for $O_1, \cdots, O_K$ via the set of mappings $\{\phi_i | i = 1, K\}$.

Definition 4.41. (Adapted and extended from [Bonatti *et al.*, 2003]) Let X be a data set (table) in the integrated domain. We define the $[X]_O = \{X, S, O\}$ inductively as follows:

- *Primitive Data Set:* If X is a data set corresponding to a data source D_i whose schema is $S_i = (A_1 : \tau_1, \cdots, A_n : \tau_n)$, then $[X]_O = (\phi_i(D_i), S, O)$, where $S = (A_1 : \phi_i(\tau_1), \cdots, A_n : \phi_i(\tau_n))$. In this case, X is always well-typed.

- *Cast:* If the data set X is obtained from a data set X' (where $[X']_O = (D, S', O)$) by converting the schema S' to a different schema S, denoted $X = (S)X'$, then $[X]_O = (S'2S(D), S, O)$. X is well-typed if S and S' have the same number of attributes, the conversion function $S'2S$ exists and X' is well-typed.

- *Projection:* If X is a data set obtained by applying PROJECT operator to an existing data set X', denoted $X = \Pi_{A_{i_1}, \cdots, A_{i_k}}(X')$, $(1 \leq i_j \leq n$ for $1 \leq j \leq k)$ and if $[X']_O =$

$(D', (A_1 : \tau_1, \cdots, A_n : \tau_n), O)$, then $[X]_O = (D, S, O)$, where D is the standard projection of D' onto $A_{i_1}, \cdots, A_{i_k}$ and $S = (A_{i_1} : \tau_{i_1}, \cdots, A_{i_k} : \tau_{i_k})$. X is well-typed if X' is well-typed.

- *Selection:* If X is a data set obtained by applying SELECT operator to an existing data set X', denoted $X = \sigma_c(X')$, and if c is a selection condition in the context of $[X']_O = (D', S, O)$, then $[X]_O = (D, S, O)$, where $D = \{t \in D' | (D', S, O), t \models c\}$. X is well-typed if X' and c are well-typed.

- *Cross Product:* If X is a data set obtained by applying CROSS PRODUCT operator to two existing data sets X_1 and X_2, denoted $X = X_1 \times X_2$ and $[X_i]_O = (D_i, S_i, O)$ (for i=1,2), then $[X]_O = (D, S, O)$, where D is the standard cross product of D_1 and D_2 and S is the concatenation of S_1 and S_2. X is well-typed if X_1 and X_2 are well-typed and S_1 and S_2 have no common attribute.

- *Join:* If X is a data set obtained by applying $JOIN$ operator to two existing data sets X_1 and X_2, we can write the join as the composition of the SELECT and CROSS PRODUCT operators, denoted $X = X_1 \bowtie X_2 = \sigma_c(X_1 \times X_2)$, and thus, we can use the previous definitions for SELECT and CROSS PRODUCT to define $[X]_O$. X is well-typed if the results of the SELECT and CROSS PRODUCT are well-typed.

- *Set Operations:* If $X = X_1$ op X_2, where op $\in \{\cup, \cap, -\}$, and if $[X_i]_O = (D_i, S_i, O)$ $(i = 1, 2)$, and S_1 and S_2 have a least common superschema S, then $[X]_O = (D, S, O)$, where D is the standard result of $S_1 2S(D_1)$ op $S_2 2S(D_2)$. X is well-typed if X_1 and X_2 are well-typed and the schemas S_1 and S_2 have a least common superschema.

- *Horizontal Integration:* If $X = X_1 \amalg X_2$, and if $[X_i]_O = (D_i, S_i, O)$ $(i = 1, 2)$, and S_1 and S_2 have a least common superschema S, then $[X]_O = (D, S, O)$, where D is the standard result of $S_1 2S(D_1) \cup S_2 2S(D_2)$. X is well-typed if X_1 and X_2 are well-typed and the schemas S_1 and S_2 have a least common superschema.

- *Vertical Integration:* If $X = X_1 \Pi X_2$, and if $[X_i]_O = (D_i, S_i, O)$ $(i = 1, 2)$, and S_1 and S_2 have at least one common column (e.g., *id*), then $[X]_O = (D, S, O)$, where D is

the result of $S_1 2S(D_1) \cdot S_2 2S(D_2)$ (where $\cdot$ means the concatenation by omitting the repetitive columns). X is well-typed if X_1 and X_2 are well-typed.

Theorem 4.42. *For all data sets X over $(D_1, S_1, O_1), \cdots, (D_K, S_K, O_K)$ and all integration ontologies O of $O_1, \cdots, O_K$, $[X]_O$ is an ontology-extended data set in the integrated domain (i.e., a table satisfying the conditions (1) and (2) in the Definition 4.29).*

Proof. Follows from the definitions above. □

4.2.2 Ontology-Extended Statistical Operators

Definition 4.43. For an ontology-extended data set X, we define an ontology-extended statistic $[f(X)]_O$ inductively as follows:

- If X is a data set, such that $[X]_O = \{D, S, O\}$, and f is an aggregate operator $f \in \{AVG, COUNT, DIST, MIN, MAX\}$ or a specialized operator $f \in \{SVM, DT, NN, NB, kNN\}$, then $[f(X)]_O = \{f(D), S', O\}$, where $f(D)$ is the result of applying the operator f to the data source D (presented as a table) and S' represents its corresponding schema. $f(X)$ is always well-typed.

- Let $f(X_1)$, $f(X_2)$ be the results of applying an aggregate or a specialized operator f to the data sets X_1 and X_2, respectively, where $[f(X_i)]_O = \{f(D_i), S'_i, O\}$ for $i = 1, 2$ are defined as above. Let $X = X_1$ op X_2 and $f(X) = g(f(X_1), f(X_2))$. If S'_1 and S'_2 have a least common superschema S', then $[f(X)]_O = \{g(f(D_1), f(D_2)), S', O\}$. $f(X)$ is well-typed if $f(X_1)$ and $f(X_2)$ are well-typed and the schemas S'_1 and S'_2 have a least common superschema. For example, if f is the $COUNT$ operator, then g is the compositional operator $+$; if f is SVM then g is $\cup$; if f is a specialized operator, g can be $VOTE$ compositional operator etc.

Theorem 4.44. *For all data sets X over $(D_1, S_1, O_1), \cdots, (D_K, S_K, O_K)$ and all integration ontologies O of $O_1, \cdots, O_K$, $[f(X)]_O$ is an ontology-extended statistic.*

Proof. Follows from the definitions above. □

4.3 Semantic Heterogeneity and Statistical Queries

Before we can develop methods to answer statistical queries from semantically heterogeneous data, it is useful to explore what it means to answer a statistical query in such a setting. In what follows, we will consider *is-a* hierarchies over attributes. We illustrate some of the issues that have to be addressed using the weather example. Thus, we assume there exist two data sources D_1 and D_2 with the associated ontologies O_1 and O_2 and a user is interested in analyzing the data from D_1 and D_2 from his perspective, which corresponds to the ontology O_U and a set of interoperation constraints IC. Suppose D_1 contains 10 instances of *Rainy* days and 30 instances of *Snowy* days. The data source D_2 contains 10 instances of *LightRain* days, 20 instances of *HeavyRain* days, 10 instances of *LightSnow* days and 10 instances of *HeavySnow* days.

A statistical query q^{O_U} is posed to the two data sources based on the ontology O_U: What fraction of the days are *Rain* days? After performing the necessary mappings ($Rainy : O_1 \rightarrow Rain : O_U$, $Rain : O_2 \rightarrow Rain : O_U$), the answer to this query can be computed in a straightforward way as the ratio of the number of *Rain* days (20+10+20=50) divided by the total number of days (100) yielding an answer of 0.5.

Now consider another query r^{O_U} (also based on the ontology O_U): What fraction of days are *HeavyRain* days? The answer to this query is not as straightforward as the answer to the previous query q_{O_U}. This is due to the fact that the quantification of rain for the days in data source D_1 is only *partially specified* [Zhang and Honavar, 2003] with respect to the ontology O_U. Consequently, we can never know the precise fraction of days that are *HeavyRain* days based on the information available in the two data sources. However, if it is reasonable to assume that the data contained in both D_1 and D_2 are drawn from the same universe (i.e., can be modeled by the same underlying distribution), we can estimate the fraction of days that are *HeavyRain* days in the data source D_1 based on the fraction of *Rain* days that are *HeavyRain* days in the data source D_2 (i.e., 20 out of 30) and use the result to answer the query r^{O_U}. Under the assumption that the samples of days in D_1 and D_2 can be modeled by the same distribution, the estimated number of *HeavyRain* days in D_1 is given by $\left(\frac{20}{30}\right)(20) = \left(\frac{40}{3}\right)$.

Hence, the estimated number of *HeavyRain* days in D_1 and D_2 is $\left(\frac{40}{3}\right) + 20 = \left(\frac{100}{3}\right)$. Thus, the answer to the query r^{O_U} is $\left(\frac{100}{3}\right)\left(\frac{1}{100}\right) = \frac{1}{3}$.

While the assumption that the data sources under consideration can be modeled by the same underlying distribution may be reasonable in some cases, in other cases, alternative assumptions may be justified. For example, some users might want to assume that the precise amount of rain in data source D_1 cannot reasonably be estimated on the basis of the rain distribution of the days in data source D_2 and hence require that the answer to query r^{O_U} be based only on the data in D_2, yielding an answer of 20 out of 100 or 0.2. An alternative would be to assume that *Rain* days in data source D_1 are equally likely to be *LightRain* or *HeavyRain* yielding an answer 0.3 (30 out of 100) to query r^{O_U}.

Note that the answer to query q^{O_U} is completely determined by the ontologies O_1, O_2, O_U, the mappings shown in Tables 4.3, 4.4 and the data available in the data sources D_1 and D_2. However, answer to the query r^{O_U} is only partially determined by the ontologies O_1, O_2, O_U, the mappings shown in Tables 4.3, 4.4 and the data available in the data sources D_1 and D_2. In such cases, answering statistical queries from semantically heterogeneous data sources requires the user to supply not only the mappings between ontologies associated with the data sources and his or her ontology, but also additional assumptions of a statistical nature (e.g., that data in D_1 and D_2 can be modeled by the same underlying distribution). The validity of the answer returned depends on the validity of the assumptions and the soundness of the procedure that computes the answer based on the supplied assumptions.

We assume that the user has the option to choose between two ways of answering queries from partially specified data with respect to his/her ontology: first option, the data in each of the distributed data source is modeled by the same underlying distribution; second option, the possible values of attributes that are partially specified are equally likely.

In the next section, we assume that we know how to solve the semantic heterogeneity problem (see Section 4.1) and we will show how we can use approaches to learn classifiers from partially specified data [Zhang and Honavar, 2003; 2004] to design algorithms for learning from semantically heterogeneous distributed data.

4.4 Algorithms for Learning Classifiers from Heterogeneous Distributed Data

We assume that all the ontologies involved (user ontology and data sources ontologies) consist of *is-a* hierarchies over the set of attributes (a.k.a., attribute value taxonomies or AVTs) (see Figure 4.7).

Definition 4.45. If A is an attribute in D, its corresponding *attribute value taxonomy*, $H(A)$ is a tree rooted at A. We denote by $Nodes(A)$ the set of nodes of the AVT associated with the attribute A. The set of leaves in the tree, $Leaves(H(A))$, corresponds to the set of primitive values of A. The internal nodes of the tree correspond to the abstract values of the attribute A. The arcs of the tree correspond to *is-a* relationships between attribute values that appear in adjacent levels in the tree. The set of abstract values at any given level in the tree $H(A)$ form a partition of the set of values at the next level (and hence, a partition of the set of primitive values of A).

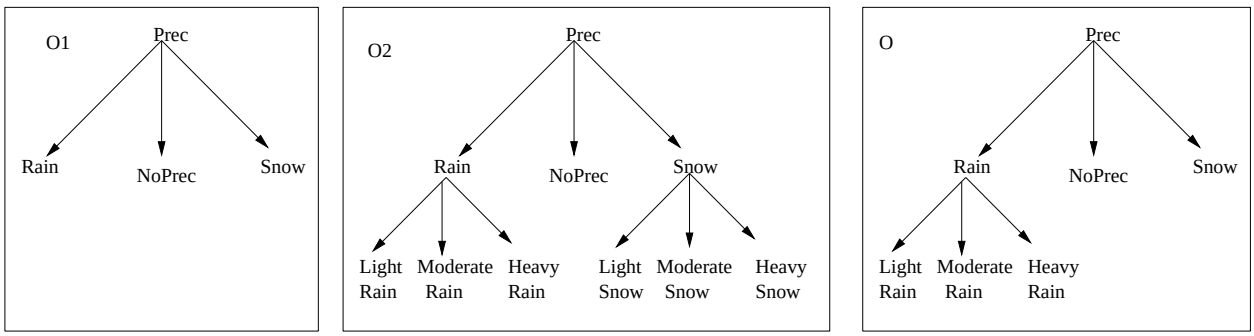

Figure 4.7 The AVTs corresponding to the *Prec* attribute in the ontologies O_1, O_2 and O_U, associated with the data sources D_1 and D_2 and a user, respectively (after the names have been matched)

Definition 4.46. [Haussler, 1988] A *cut* $Z(H(A))$ of an AVT $H(A)$ is a subset of nodes in $H(A)$ satisfying the following two properties:

- For any leaf $v \in Leaves(H(A))$, either $v \in Z$ or v is a descendent of a node $n \in Z$.
- For any two nodes $n_1, n_2 \in Z$, n_1 is neither a descendent nor an ascendent of n_2.

Cuts through an AVT $H(A)$ correspond to partitions of $Leaves(H(A))$.

Example 4.47. The cut corresponding to $\{Rain, Snow, NoPrec\}$ in the AVT associated with the attribute $Prec$ in the ontology O_2 in Figure 4.7, defines a partition of the primitive values of the $Prec$ attribute as follows: $\{LightRain, ModRain, HeavyRaib\}, \{NoPrec\}$, $\{LightSnow, ModSnow, HeavySnow\}\}$.

If $S = (A_1 : \tau_1, A_2 : \tau_2, \cdots, A_n : \tau_n)$ is the schema of a data source D having an ontology O, then O can be written as $O = \{H_1(A_1), H_2(A_2), \cdots, H_n(A_n)\}$, where $H_i(A_i)$ is the AVT corresponding to the attribute A_i.

Definition 4.48. A *cut* $Z(O)$ of an ontology $O = \{H_1(A_1), H_2(A_2), \cdots, H_n(A_n)\}$ is defined as $Z(O) = \{Z(H_1(A_1)), Z(H_2(A_2)), \cdots, Z(H_n(A_n))\}$.

Let $(D_1, S_1, O_1), \cdots, (D_K, S_K, O_K)$ be K ontology-extended data sources and O_U a user ontology. Let $Z(O_1), \cdots, Z(O_K)$ be the levels of abstraction (cuts) at which the instances are specified in the data sources $D_1, \cdots, D_K$, respectively and $Z(O_U)$ a *learning cut* through the user ontology defining the level of abstraction at which the learning needs to be done. When learning from $D_1, \cdots, D_K$ using the user ontology O_U, the name and type heterogeneity problems are solved once valid mappings between data source ontologies and user ontology have been specified. However, we still encounter problems as those described in the previous section. More precisely, having different ontologies at different data sources implies that the instances to be classified could be specified at different levels of precision with respect to a user ontology.

Definition 4.49. Let $x = (v_{A_1}, \cdots, v_{A_n}) \in D_j$ be an instance in D_j. We say that x is:

- a *completely specified instance* if for all $1 \leq i \leq n$, the correspondent of v_{A_i} in O_U belongs to the user level of abstraction $Z(O_U)$.

- a *partially specified instance* if there exist at least one attribute value v_{A_i} for which the corresponding value in $Z(O_U)$ does not belong to the user level of abstraction $Z(O_U)$. This value can be *under-specified* if its correspondent in the user ontology is above the

learning cut, or *over-specified* if its correspondent in the user ontology would be below the learning cut (but it actually does not exist). An attribute is *under-specified* if it has *under-specified* values, and it is *over-specified* if it has *over-specified* values.

Example 4.50. Assume that the instances in the data source D_1 are specified in terms of $Rain$, $NoPrec$ and $Snow$. The instances in the data source D_2 are specified in terms of $LightRain$, $ModerateRain$, $HeavyRain$, $NoPrec$, $LightSnow$, $ModerateSnow$, $HeavySnow$. Assume that according to the user level of abstraction the instances have to be specified in terms of $LightRain$, $ModerateRain$, $HeavyRain$, $NoPrec$ and $Snow$. We can see that in this case, the instances in D_1 are under-specified, while the instances in D_2 are over-specified. Thus, $Rain$ is an under-specified value of the attribute $Prec$ in D_1, while $LightSnow$, $ModerateSnow$, $HeavySnow$ are over-specified values of the attribute $Prec$ in D_2.

One way to deal with the under- or over-specification problems is to replace the original data set with a new data set where the values of the attributes are at the right level of specification, given the user level of abstraction, and then apply the algorithms for learning from distributed data described in Chapter 3. In principle, this could be easily done when an attribute is over-specified: we replace the over-specified value with a higher level ancestor in the corresponding AVT (specifically, with the ancestor that has the same level of abstraction as the value in the user AVT). When an instance is under-specified, we replace the original instance with a new instance having the right level of specification, according to the user preference. Thus, the user specifies how under-specified values should be filled in: by assuming that all the data is modeled by the same underlying distribution or by assuming uniform distribution for the data.

Although we can, in principle, generate a new data set having the right level of specification, this is not always possible in a distributed environment where data sources are autonomous. We will show that for some learning algorithms we can gather the sufficient statistics corresponding to the transformed data sets (having the right level of abstraction) without doing the transformation explicitly.

Let $A_1(O_U), \cdots, A_n(O_U)$ be the user attributes with respect to a data domain and $O_U =$

$\{H_1(A_1), \cdots, H_n(A_n)\}$ the user ontology associated with these attributes. Let $v_{A_1}(O_U), \cdots, v_{A_n}(O_U)$ be a learning cut through the user ontology (note that $v_{A_i}(O_U) \subseteq H_U(A_i)$ could be a set of values of the attribute $A_i(O_U)$). If the data are horizontally distributed, then each data source D_j contains an attribute $A_i(O_j)$ that maps to $A_i(O_U)$. If the data are vertically distributed, then for each attribute $A_i(O_U)$ there exists a data source D_j that contains an attribute $A_i(O_j)$ that maps to $A_i(O_U)$.

4.4.1 Naive Bayes Classifiers from Heterogeneous Data

The algorithm for learning naive Bayes classifiers from horizontally (vertically) distributed heterogeneous data sources is similar to the algorithm for learning naive Bayes classifiers form horizontally (vertically) distributed homogeneous data sources. As opposed to this case, in the case of heterogeneous data sources:

First the set of mappings is used to find the correspondents of the user attributes in the distributed data sources (e.g., $A_i(O_j) \rightarrow A_i(O_U)$) and also to resolve the syntactic and semantic mismatches between the correspondent attributes.

Second, for each attribute value $v \in v_{A_i}(O_U)$ in the user cut, we compute the counts at a particular data source D_j that contains that attribute, as follows:

- If v is over-specified in D_j, then we recursively propagate up the counts from its children in $H_i(D_j)$ to v, until all the children are specified in D_j (primitives). For example, in Figure 4.7, to compute the counts in D_2 corresponding to $Snow$, we compute the counts for $LightSnow$, $ModerateSnow$, and $HeavySnow$ and we add them up.

- If v is under-specified in D_j, we can treat it as a missing value and thus we reduce our problem to the problem of filling in missing values. Depending on the user preference, one of the following approaches can be used for that:

  - If the user assumes uniform distribution over the attribute values, then the counts are propagated down uniformly from a parent to its children. For example, in Figure 4.7, if there are 9 instances in D_1 for which the attribute $Prec$ takes value

Rain, then according to the user assumption, we can infer that that there are 3 instances for each of the values $LightRain$, $ModerateRain$ and $HeavyRain$.

– If the user assumes that all the data are coming from the same distribution, we can estimate this distribution based on a data set where the values are specified, and then propagate down the counts based on that distribution in a data set where the values are under-specified. For example, if there are 8 instances in D_1 for which $Prec$ takes value *Rain* and if the distribution over the values $LightRain$, $ModerateRain$, $HeavyRain$ is $(25, 50, 25)$, then we can infer that there are 2 instances for which $Prec = LightRain$, 4 instances for which $Prec = ModerateRain$ and 2 instances for which $Prec = HeavyRain$.

Once the counts are estimated this way, the algorithm works as in the homogeneous distributed data case. Thus, we can see that we do not need to explicitly construct data sets where all the instances are completely specified, as the counts can be computed implicitly.

4.4.2 Decision Tree Induction from Heterogeneous Data

The algorithm for learning decision trees from horizontally (vertically) distributed heterogeneous data sources is similar to the algorithm for learning decision trees form horizontally (vertically) distributed homogeneous data sources. As the sufficient statistics that need to be computed in the case of the decision tree algorithm are also counts, they can be computed similar to the way we compute the counts for naive Bayes, after all the mappings are performed. Thus, there is no need to explicitly construct the equivalent data sets where all the instances are completely specified.

4.4.3 Support Vector Machines from Heterogeneous Data

In the case of the Support Vector Machines algorithm, we showed that if the data are horizontally distributed, the sufficient statistics are given by the support vectors (if we iterate a few times through the data) or the points that determine the convex hull. In either case, an optimization problem that involves all data needs to be solved in order to find the sufficient

statistics. Because of that, it is not possible to compute the sufficient statistics without effectively constructing new data sets, where all the under-specified or over-specified values are filled in or abstracted as follows:

- If v is over-specified in D_j, let $u_1, \cdots, u_k$ be the over-specifications of v in D_j. We replace every instance in D_j with a new instance in which any occurrence of the values $u_1, \cdots, u_k$ is replaced by the values v. For example, for D_2 in Figure 4.7, we replace $LightSnow$, $ModerateSnow$, and $HeavySnow$ with $Snow$ in any instance from D_j.

- If v is under-specified in D_j, we have to fill it in according to the user preference. Thus, one of the following approaches can be used to fill in under-specified values:

  - If the user assumes uniform distribution over the attribute values, we replace v's correspondent in D_j randomly with one of v's children (the probability that any child is used is 1/(number of children)). If the new value is a primitive value in O_U, we are done, otherwise the same procedure is repeated until all the values are primitive.

  - If the user assumes that all the data are coming from the same distribution, we can estimate this distribution based on a data set where the values are specified, and then replace v's correspondent in D_j with one of v's children according to the estimated distribution. If the new value is a primitive value in O_U, we are done, otherwise the same procedure is repeated until all the values are primitive.

  For example, for the data set D_1 the value $Rain$ of the attribute $Prec$ is under-specified. It needs to be replaced with one of the children of the $Rain$ value in O_U (i.e., $LightRain$, $ModerateRain$ or $HeavyRain$) according to the desired distribution.

Another method to construct a data set where all the instances are specified, based on a data set containing over- or under-specified values, is called *propositionalization*. According to this method, if $v_{A_1}(O_U), \cdots, v_{A_n}(O_U)$ is the learning cut, any instance in a data source D_j is replaced with a new instance that has as many boolean attributes as values in the learning cut. Any of these new attributes can take one of the values *True*, *False*, or *Missing*, as follows:

- If an attribute A, corresponding to the value v in the learning cut, is at the right level of specification in D_j, then: if A appears in an instance D_j, it takes value *True* in the new instance. If it A does not appear in an instance, it takes value *False* in the new instance.

- If an attribute A, corresponding to the value v in the learning cut, is under-specified in D_j, then it takes value *Missing* in any new instances.

- If an attribute A, corresponding to the value v in the learning cut, is over-specified in D_j, then it takes value *True* in any new instances.

Observation 4.51. Gradient-based variants of SVM algorithm could be transformed into algorithms for learning from partially specified distributed data without constructing the data sets explicitly by using an approach similar to the approach for learning threshold functions from heterogeneous data (see Section 4.4.4).

4.4.4 Learning Threshold Functions from Heterogeneous Data

We saw that in the case of learning threshold functions, the weight w represents the sufficient statistics. This weight is updated at each step of the algorithm based on the current example in the case horizontally distributed data (when data are vertically the weight is updated once for each data source, using a quantity computed by visiting the instances one by one). Because of the incremental nature of these algorithms, it is not necessary to explicitly construct new data sets containing completely specified instances. Instead when needed, each over- or under-specified instance is transformed *on the fly* into the corresponding specified instance using one of the methods described in Section 4.4.3. Thus, after resolving name and type heterogeneity problems, the distributed algorithms described in Chapter 3 can be used unchanged, except that the weight is updated based on the (implicitly) transformed instances.

4.4.5 k-Nearest Neighbors Classifiers from Heterogeneous Data

Similar to the algorithms for learning threshold functions, the k-NN classifiers are incremental with respect to instances, meaning that they compute distances (i.e., sufficient

statistics) by processing the examples one by one. Thus, both horizontally and vertically distributed k-NN classifiers can be used as they are for learning from horizontally and vertically distributed heterogeneous data sources. The only difference is that the distances are computed not using the original instances, but using (implicitly) transformed instances, as in the case of learning threshold function. Thus, this is another example of an algorithm where the data are not explicitly transformed from partially specified data to completely specified data.

4.5 Summary and Discussion

In this Chapter, we showed how the approach for learning from distributed data sources can be extended to yield an approach for learning from heterogeneous data sources. To do that, we defined ontologies, user perspective and the integration of a set of ontologies from a user perspective. We associated an ontology with each data source. In this setting, answering statistical queries from ontology-extended data sources reduces to extending operators with ontological information, so that their invocation results in well-typed data sets (tables) or statistics over data sets. We showed that learning from heterogeneous data sources can be reduced to learning from partially specified data in the presence of AVT's [Zhang and Honavar, 2003]. We used the approach in [Zhang and Honavar, 2003] together with the approach in Chapter 3 [Caragea *et al.*, 2004d] to design algorithms for learning Naive Bayes, Decision Trees, Perceptron, SVM and k-NN classifiers from heterogeneous distributed data.

Our definition of ontology-extended data sources was inspired by a similar definition for ontology-extended relational algebra introduced in [Bonatti *et al.*, 2003]. The authors in [Bonatti *et al.*, 2003] associate a graph with each hierarchy. In their setting, the user defines a set of mappings between different hierarchies in the system and a set of interoperation constraints. The mappings are used to merge all the individual graph hierarchies into an overall graph hierarchy. An integration hierarchy is given by a canonical hierarchy which consists of all strongly connected components in the graph hierarchy. An integration hierarchy is valid if it satisfies a set of interoperation constraints and order preservation property.

As opposed to [Bonatti *et al.*, 2003], we define a user perspective as consisting of a user ontology and a set of interoperation constraints. We present a simple algorithm for coming up with mappings between data source ontologies and a user ontology based on interoperation constraints and an algorithm for checking that these mappings are valid.

Our approach is more general that the approach in [Bonatti *et al.*, 2003] because users can impose their own perspective over a set of data sources. It is also more general in the sense that our data sources can be in any format (e.g., flat files, relational databases, web pages etc.) and thus the set of operators used to retrieve data or statistics is an extension of the relational operators.

Our results are similar to the results in [McClean *et al.*, 2002] in terms of the flexibility achieved by giving the user the possibility to specify his/her own ontology. However, their framework assumes that there exists metadata, in terms of mappings between ontologies, in the system, while we give the user the possibility to specify how he/she wants to use the existent data, by specifying a set of interoperation constraints that relates data of interest. Another strength of our approach comes from the ability to deal with type heterogeneity (by using conversion functions, e.g. $C \rightarrow F$), not only with name ($Temp \rightarrow Temperature$) and level of abstraction heterogeneity (e.g. $LightRain \rightarrow Rain$).

The approach to learning from ontology-extended data sources is similar to the approach in [Zhang and Honavar, 2003], where AVT's are associated with the attributes in a data set and the level of abstraction which gives the best accuracy is sought. In our case, we assume the level the abstraction is given by the user. This level defines a level of abstraction for each data source ontology, which results in some attributes being over-specified while others might be under-specified, hence the connection with learning from partially specified data. We can envision scenarios where there is no user predefined level of abstraction, in which case we would iterate through successive user levels of abstraction as in [Zhang and Honavar, 2003; 2004] and the one that gives the best accuracy is chosen.

Pathak *et al.* [2004] developed ontology-extended workflow components and semantically consistent methods for assembling such components into complex ontology-extended component-based workflows. The result is a sound theoretical framework for assembly of se-

mantically well-formed workflows from semantically heterogeneous components. In this case, there is no integration hierarchy for all the ontologies associated with components in the workflow, as some of them may be unrelated. Instead an integration ontology is found for every set of ontologies corresponding to neighboring (source, target) components.

Bromberg *et al.* [2004] defined the problem of multi-agent data mining, which is an extension to our framework for learning from distributed data. In multi-agent data mining, the agents have limited resources and are self-interested, but they can achieve their goals by communicating and exchanging information with other self-interested agents. Thus, mechanisms for knowledge production and coordination, similar to those in economics, need to be developed. We assume that there are hundreds of agents in such a framework, so one agent cannot communicate with all the agents in the system but just with a small subset of agents. One natural extension to the framework in [Bromberg *et al.*, 2004] is to associate ontologies with each agent in the system. As in the case of workflow components, we do not have an overall integration ontology, but we can define integration ontology for the neighborhood of an agent.

5 SUFFICIENT STATISTICS GATHERING

In the previous chapters we have seen that learning from distributed heterogeneous data can be reduced to identifying sufficient statistics for the learning algorithm, gathering the sufficient statistics from heterogeneous data and generating the classifier using these sufficient statistics. We have identified sufficient statistics for a representative class of learning algorithms and showed how they can be used to generate the classifiers. We have also developed the tools needed to gather sufficient statistics by introducing a statistical query language and extending data sources and query operators with ontologies. In this chapter we will show how we can design a system for gathering sufficient statistics (i.e., answering statistical queries) from distributed heterogeneous autonomous data sources.

5.1 System Architecture

The architecture of the system for gathering sufficient statistics from distributed heterogeneous autonomous data sources is similar to the architecture of many heterogeneous database systems [Haas *et al.*, 1997; Tomasic *et al.*, 1998; Garcia-Molina *et al.*, 1997; Chang and Garcia-Molina, 1999; Rodriguez-Martinez and Roussopoulos, 2000], etc. Figure 5.1 shows this architecture. It consists of *servers* (data sources) and *clients* (learning algorithms) that are registered with a *central resource repository*. A set of *iterators* used to access and retrieve information from data sources and a *user perspective* of the system are also registered with the central resource repository. A *query answering engine* (query optimizer), which acts as a middleware between clients and servers, is used to answer statistical queries from autonomous semantically heterogeneous data sources, under a variety of constraints and assumptions motivated by application scenarios encountered in practice. The query answering engine has access

to the data sources in the system through *data access modules* (DA), which are invocations of iterators.

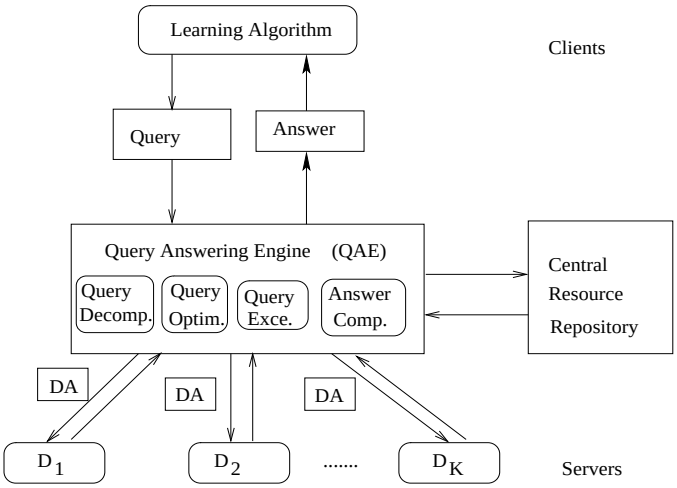

Figure 5.1 The architecture of a system for gathering sufficient statistics from distributed heterogeneous autonomous data sources

In what follows, we will describe the central resource repository (Section 5.2) and the query answering engine (Section 5.3). We will show how the problem of gathering sufficient statistics from distributed heterogeneous autonomous data sources can be formulated as a planning problem (Section 5.4.1) and present a planning algorithm for solving this optimization problem (Section 5.4.2).

5.2 Central Resource Repository

We assume that all the information available in the system is registered with a *central resource repository*. The central resource repository has four main components *data catalog*, *algorithm catalog*, *iterator repository* and *user perspective*, that will be described below (see Figure 5.2).

First, in any distributed environment that we consider there exist several *data sources* that store interrelated data. Every data source available in the system registers with the *data catalog* component of the central resource repository. When registering, both the location (network address, i.e., URI) of the data source and the description of the data source are provided. The description of a data source D consists of the data source schema, S, its

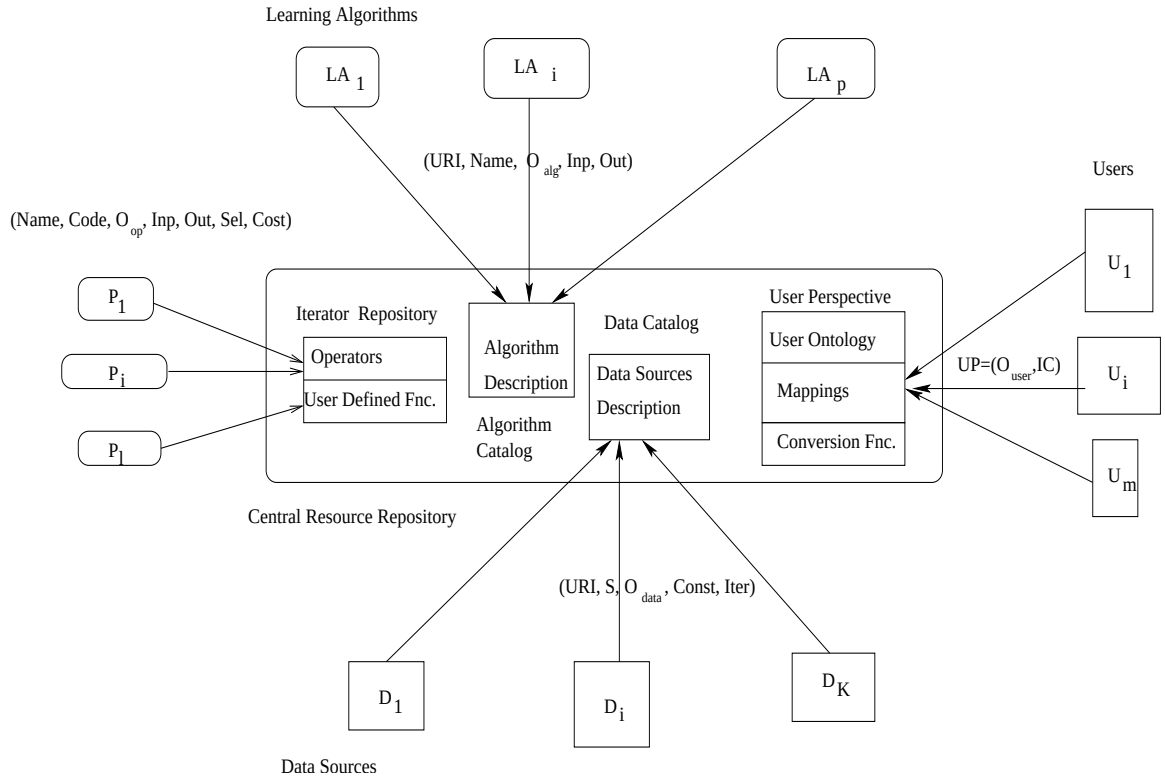

Figure 5.2 Central resource repository: data sources, learning algorithms, iterators and users registration

ontology, O_{data}, the set of constraints, $Cons$, imposed by that particular data source and also a wrapper consisting of iterators, $Iter$, that can be directly executed on the data source (e.g., calculation of counts of data source instances that satisfy certain constraints on the values of some of the attributes). Thus, a data source D registers as a tuple $(URI, S, O_{Data}, Cons, Iter)$. New data sources can be easily added to the system by registering them with the query answering engine by specifying such a tuple.

Besides data sources that store data, in a distributed system there exist several *learning* or *information extraction algorithms* that can be used to learn or extract information from the available data sources. Similar to data sources, the learning algorithms register with the *algorithm catalog* in the resource repository. For an algorithm, the location (URI) of its code and its description, consisting of $Name$ of the algorithm, its associated ontology, O_{Alg}, inputs, Inp, and outputs, Out, are provided. New learning algorithms can be added to the system by specifying a tuple like $(URI, Name, O_{Alg}, Inp, Out)$.

All the *iterators* that can be used in the process of answering queries are registered with

an *iterator repository*. These iterators are implementations of primitive and statistical operators, or user defined functions. Besides code, their names, information about their inputs and outputs, as well as an ontology over their names is recorded in the iterators repository. If available, their selectivity factors and their associated costs (w.r.t. number of instances, number of attributes etc.) if executed locally or globally are also recorded. In the paper [Krishnaswamy *et al.*, 2002] a suite of techniques for estimating the computation and communication costs of distributed data mining are presented. Thus, an iterator registers as a tuple $(Name, Code, O_{Op}, Inp, Out, Sel, Cost)$. New iterators can be added to the system by registering the corresponding information with the iterators repository. The existing ones can be easily updated by re-submitting this information.

When a user wants to use the system for learning or just for information extraction, the user is given access to all the data sources, algorithms and iterators available in the system and he can build simple *workflows* like those in Figure 5.3 using some of these resources according to his needs. For the system to be able to execute this workflow, the current user or an expert user needs to provide the central resource repository with the user perspective UP consisting of an ontology O_{user} and a set of interoperation constraints IC from his ontology to the ontologies of the data sources, algorithms and iterators that are involved in the user workflow. The set of interoperation constraints is used by the system to come up with mappings from the user ontology to other ontologies involved in the user workflow, as described in Chapter 4. Once a set of mappings is found, the user can inspect these mappings, delete, add or modify them, and also associate conversion functions with the final set of mappings (either new conversion function or predefined conversion functions). All this information is stored in the *user perspective* component of the *central resource repository*. According to the semantic imposed by the user ontology, the workflows are internally translated by the system into more specific workflows, such as those in Figure 5.4.

All the resources in the system that register with the *central resource repository* are described in a RDF file (Resource Description Framework) [RDF, 1995], which is an XML-based technology used to specify metadata for resources available in a networked environment. Figure 5.5 shows the RDF file of a data source (Prosite) described by *name*, *URI*, *schema* and

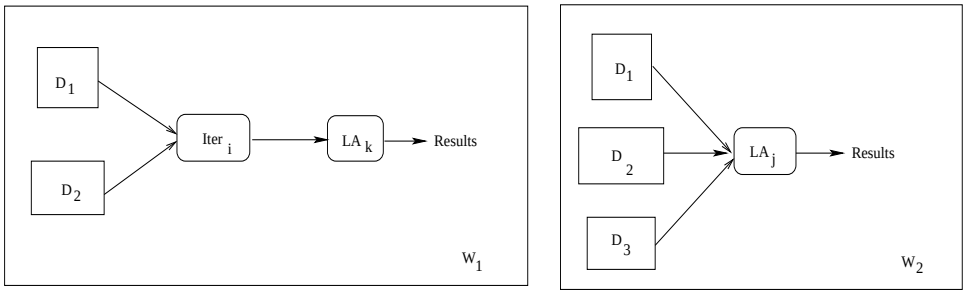

Figure 5.3 Simple user workflow examples

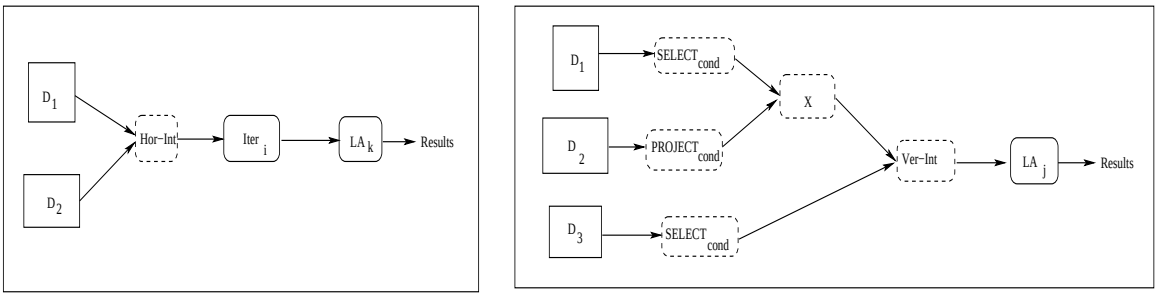

Figure 5.4 Internal translation of the workflows in Figure 5.3 according to the semantic imposed by the user ontology

operators allowed by the data source.

5.3 Query Answering Engine

In the process of workflow execution, queries are sent to a *query answering engine* (QAE), which plays the role of a mediator (*middleware*) between the learning algorithm or the user that formulates the query (clients) and data sources (servers) (see Figure 5.1). The query answering engine has information about the resources available in the systems and the constraints and assumptions imposed by them, through the central resource repository. Besides, it can access the distributed data sources and extract information from them according to their constraints, through data access modules (DA), which are invocations of iterators.

We design the query answering engine by adapting the middleware in [Rodriguez-Martinez and Roussopoulos, 2000] according to the needs of the learning algorithms that we consider. Thus, we assume that the middleware is *self extensible*, meaning that if a particular application needs a specific functionality that is not provided by a remote data source (visited in the query answering process), then the middleware itself deploys the code that realizes that functionality

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
    xmlns:INDUS="http://pierce.cs.iastate.edu/7080" >

    <rdf:Desc rdf:about="http://Indus/Database/">
    <INDUS:Database> Prosite </INDUS:Database>
    <INDUS:URI> ftp://us.expasy.org/databases/prosite/release </INDUS:URI>
    <INDUS:Schema>
        <rdf:Bag>
            <rdf:li parseType="Resource" >
                <INDUS:A_Name> ID </INDUS:A_Name>
                <INDUS:A_Type> Integer </INDUS:A_Type>
                <INDUS:A_Desc>Indicates ID of the Protein </INDUS:A_Desc>
            </rdf:li>
            <rdf:li parseType="Resource" >
                <INDUS:A_Name> AC </INDUS:A_Name>
                <INDUS:A_Type> String </INDUS:A_Type>
                <INDUS:A_Desc>Accession number of the Protein </INDUS:A_Desc>
            </rdf:li>
        </rdf:Bag>
    </INDUS:Schema>
    <INDUS:Operators>
        <rdf:Bag>
            <rdf:li parseType="Resource" >
                <INDUS:F_Name> Sort </INDUS:F_Name>
                <INDUS:F_Desc> Sorts the records by Date </INDUS:F_Desc>
            </rdf:li>
            <rdf:li parseType="Resource" >
                <INDUS:F_Name> Count </INDUS:F_Name>
                <INDUS:F_Desc> Give counts on some attribute </INDUS:F_Desc>
            </rdf:li>
        </rdf:Bag>
    </INDUS:Operators>
    </rdf:Description>
</rdf:RDF>
```

Figure 5.5 Example of RDF file for a data source (Prosite) described by *name*, *URI*, *schema* and *operators* allowed by the data source

to the data source in automatic fashion, if that particular data source allows this. As a result, the remote data source achieves new capabilities, becoming able to manipulate the data of interest, and thus satisfies the need for application-specific operators at the remote sites that do not provide them [Rodriguez-Martinez and Roussopoulos, 2000; Yang *et al.*, 1998].

The query answering engine consists of four main components (see Figure 5.6): *query decomposition*, *query optimization*, *query execution*, and *answer composition*.

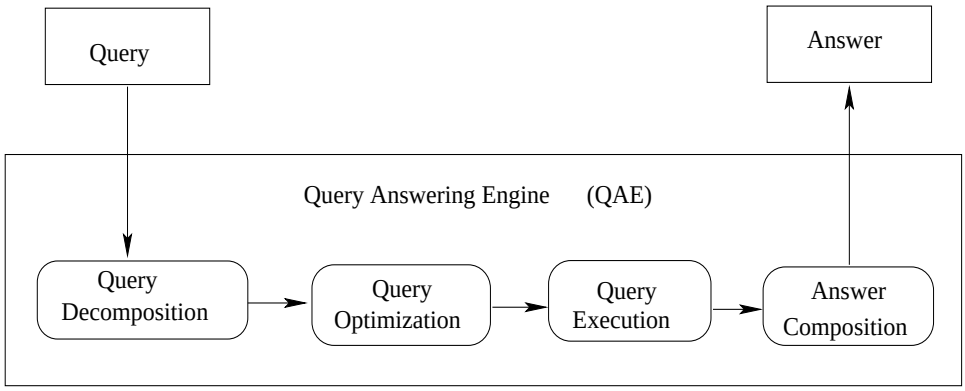

Figure 5.6 Query answering engine

The *query decomposition* component decomposes a query into sub-queries by identifying the largest possible fragments that involve a particular data source and it sends them to the query optimization component together with the specification of how to combine the results.

For each sub-query that it receives, the *query optimization* component enumerates all the possible plans (or only some of them if a heuristic is used), eliminates those that do not comply with the constraints imposed by the corresponding data source, then computes the cost of each of the remaining plans based on the cost of operators involved and chooses the best one. The optimal composition of the individual sub-query plans results in an overall best plan to the original query that needs to be answered. Once the best plan is found, it is sent to the query execution component.

The *query execution* component sends the plan corresponding to a data source to its corresponding DA. The DA invokes some iterators and returns the answer of the query with respect to that data source to the answer composition component.

The *answer composition* component puts together all the answers received from the local data sources according to the composition operators identified by the query decomposition

component and sends the final answer to the client that asked the query.

Example 5.1. Assume that there are K data sources $D_1, \cdots, D_K$ available in the system and a learning algorithm asks a statistical query Q over these data sources (e.g., $counts(class =$ "$Sunny$")). This query is decomposed into K sub-queries $Q_{D_1}, \cdots, Q_{D_K}$ (e.g., $Q_{D_i} = counts_{D_i}(class =$ "$Sunny$")) and optimal plans are found for each of these sub-queries and executed, resulting in answers $A(Q_{D_i})$. The answer $A(Q)$ is obtained by composing the answers $A(Q_{D_1}), \cdots, A(Q_{D_K})$ as follows: $A(Q) = A(Q_{D_1})$ op Q_{D_2} op $\cdots A(Q_{D_K})$ (in our *counts* example op $= +$).

There is a lot of work in the information integration community on designing query languages and rules for decomposing queries into sub-queries and composing the answers to sub-queries into answers to the initial query [Garcia-Molina *et al.*, 1997; Chang and Garcia-Molina, 1999; Knoblock *et al.*, 2001; Lu *et al.*, 1995; Levy, 1998; Draper *et al.*, 2001] etc. We rely on the results of this work for query decomposition, query execution and answer composition in our system, as there is nothing different in terms of statistical queries that we want to answer. However, we will give a more detailed description of the way we deal with the query optimization component of the query answering engine as this is where the constraints imposed by data sources and the analysis that we have done in Chapter 3 in terms of communication complexity need to be taken into account.

5.4 Query Optimization Component

Our approach to the design of query optimization component is inspired from [Rodriguez-Martinez and Roussopoulos, 2000].

5.4.1 Optimization Problem Definition

Definition 5.2. We define the *cost of an operator* Ω with respect to a data source D as follows: $cost_D(\Omega) = \alpha \cdot CompCost_D(\Omega) + \beta \cdot NetworkCost_D(\Omega) + \gamma \cdot RespTime_D(\Omega)$, where $CompCost_D(\Omega)$ is the total cost of applying the operator Ω over an input data set D, $NetworkCost_D(\Omega)$ is the total communication cost (data movement) incurred while executing the operator Ω on D, and $RespTime_D(\Omega)$ is the response time when the operator Ω is applied

on the data source D. The parameters α, β, γ are chosen according to the user preference on a certain costs.

Example 5.3. A user can set $\alpha = 0.1$, $\beta = 1$ and $\gamma = 0$ which means that the communication cost is the most important to that user, followed by a small fraction of the computation cost. The response time is ignored.

Because the time needed to execute a query is usually considerably smaller than the communication time, by optimizing the data movement, we optimize also the total time, unless the user specifies a different preference over the cost components by changing the values of the parameters α, β, γ.

Definition 5.4. Similar to [Rodriguez-Martinez and Roussopoulos, 2000], our query operators can be classified into two categories:

- *data-reducing operators* or *filters*: return answers whose volume is smaller than the volume of the data that they are applied to, and thus it is better to execute them at the remote distributed data sources (e.g., aggregate operators such as counts);
- *data-inflating operators*: return answers whose volume is larger than the volume of the data that they are applied to (e.g., some user-defined operators) and thus they should be executed at the central location where the query answering engine is located.

Thus, we ship more or less data, depending on where we execute the queries. More precisely, we can perform *data shipping*, when the data is shipped at the client site and the query is executed there, or *result shipping*, when the query is executed at the data site (via shipping the code if it is not already available) and only the results of the query are shipped at the client site. Moreover, hybrid approaches that combine data and answer shipping, depending on which one is the best at each time step, are preferable.

We use the approach in [Rodriguez-Martinez and Roussopoulos, 2000] to decide if an operator is a data-inflating operator or a data-reducing operator. This approach is based on the definitions below.

Definition 5.5. The *Volume Reduction Factor*, *VRF* for an operator Ω over a data set D is defined as

$$VRF(\Omega) = \frac{VDT}{VDA},\ (0 \leq VRF(\Omega) < \infty),$$

where VDT is the total data volume to be transmitted after applying Ω to D, and VDA is the total data volume originally in D. Thus, an operator Ω is *data-reducing* if and only if its VRF is less than 1; otherwise, it is *data-inflating*.

Definition 5.6. The *Cumulative Volume Reduction Factor*, *CVRF* for a query plan P to answer query Q over data sets $D_1, \cdots, D_n$ is defined as

$$CVRF(\Omega) = \frac{CVDT}{CVDA},\ (0 \leq CVRF(\Omega) < \infty),$$

where $CVDT$ is the total data volume to be transmitted over the network after applying all the operators in P to $D_1, \cdots, D_n$, and $CVDA$ is the total data volume originally in $D_1, \cdots, D_n$.

The intuition is that the smaller the $CVRF$ of the plan, the less data is sent over the network, and the better performance the plan provides.

5.4.2 Planning Algorithm

The query optimizer described in Figure 5.7 (similar to [Rodriguez-Martinez and Roussopoulos, 2000]) follows a dynamic programming model for query optimization. We want to find an optimal plan for the query $Q(D_1, \cdots, D_K) = Q_{D_1}\ \mathrm{op}_1\ Q_{D_2} \cdots\ \mathrm{op}_{K-1}\ Q_{D_K}$. Because for a query some operators may be executed at the remote data sources and some may be executed at the central place, we assume that each query plan has two component sub-plans, a data source sub-plan DP that is executed by data access modules corresponding to the distributed data sources and an engine sub-plan EP that is executed at the central place by the query answering engine. The first part of the algorithm described in 5.7 (1–5) construct an optimal plan for each sub-query Q_{D_i}. The construction starts with an arbitrary plan in step 3, followed by an optimization procedure that finds the optimal placement for the operators involved in the sub-query in step 4. The last part of the algorithm in Figure 5.7 (6–20) finds

```
procedure Query Optimization: Q(D_1, ··· , D_K) = Q_{D_1} op_1 Q_{D_2} ··· op_{K-1} Q_{D_K}
/* find best composition plan */
 1   for i=1 to K do
 2   {
 3       P_i ← selectPlan(D_i)
 4       optimalPlan(D_i) ← OptimalOperatorPlacement(P_i, D_i)
 5   }
 6   for  i=2 to K do
 7   {
 8       for all S ∈ {D_1, ··· , D_K} s.t. |S| = i do
 9       {
10           bestPlan ← any plan with infinite cost
11           for all D_j, S_j s.t. S = {D_j} ∪ S_j do
12           {
13               P ← composedPlan(optimalPlan(S_j), optimalPlan(D_j))
14               P ← OptimalOperatorPlacement(P, D_j)
15               if cost(P) ≤ cost(bestPlan)
16                   bestPlan ← P
17           }
18           optimalPlan(S) ← bestPlan
19       }
20  }
21  return optimalPlan({D_1, ··· D_n})
```

Figure 5.7 Query optimization (planning) algorithm

the best way to compose individual plans, using the optimal operators placement again, this time for placing the composition operators op_i with $i = 1, K-1$. The pseudocode for optimal operators placement is shown in Figure 5.8. It starts by identifying the operators in a plan

```
procedure Operator Placement(P, D)
/* find best operator placement */
 1   O ← getOperators(P, D)
 2   opDP ← initDP(P, D)
 3   opEP ← initEP(P, D)
 4   for all Ω ∈ O do
 5   {
 6       if (VRF(Ω) < 1)
 7           insert(Ω, opDP)
 8       else
 9           insert(Ω, opEP)
 10  }
 11  rank(opDP)
 12  rank(opEP)
```

Figure 5.8 Operator placement algorithm

P that can be executed at the data source D and initializes a set of operators $opDP$ that need to be executed at D and a set of operators $opEP$ that need to be executed at the central place. Then, for each of the operators in $\Omega \in P$, it places them in $opDP$ or $opEP$ in such a way that the data movement is minimized. The procedure, $rank$, ranks the operators in a set according to the metric: $rank(op) = \frac{Sel_{op}}{CompCost(op)}$, where Sel_{op} is the selectivity of the operators op defined as $Sel(op) = \frac{|output(op)|}{|input(op)|}$ (size of the output divided by size of the input) in [Hellerstein and Stonebraker, 1993] and $CompCost(op)$ represents the computational cost of op.

The best plan generated by the optimizer explicitly indicates which are the operators to be evaluated by the query answering engine and which are those to be evaluated at the remote data sites. In addition, it also indicates what code needs to be dynamically deployed to each of the participants in the query execution process.

5.5 Sufficient Statistics Gathering: Example

Suppose that we have the following resources in a system: two data sources D_1 and D_2 containing weather data as in Section 4.1.1, a Naive Bayes learning algorithm and a set of iterators for accessing data sources. The data sources D_1 and D_2 register with the central resource repository by submitting tuples such as $(URI_{D_i}, S_{D_i}, O_{D_i}, Cons_{D_i}, Iter_{D_i})$ for $i = 1, 2$, respectively. The learning algorithm sends a tuple like: $(URI_{NB}, NB, O_{NB}, Inp_{NB}, Out_{NB})$. Similarly an iterator *iter* (e.g., *counts*) submits a tuple like: $(Name_{iter}, Code_{iter}, O_{iter}, Inp_{iter}, Out_{iter}, Sel_{iter}, Cost_{iter})$ to the central resource repository. A user U that wants to use the system to learn Naive Bayes classifiers using the resources available, has to register with the system by defining his perspective over the system, i.e., an ontology O_U and a set of interoperation constraints IC from the ontologies in the system to his own ontology. The user can build a workflow like the one in Figure 5.9 (Left), which is internally translated into the workflow in Figure 5.9 (Right).

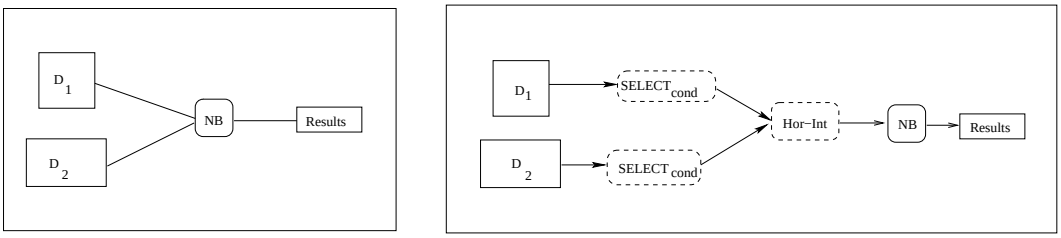

Figure 5.9 (Left) User workflow Naive Bayes example (Right) User workflow internal translation

The algorithm's input is mapped to $SelCond(D_1)$ Hor-Int $SelCond(D_2)$, where $SelCond$ is a selection condition as defined in Subsection 4.1.4 (e.g. $SELECT_{temperature \geq 0}(D_1)$), and thus the workflow can be written as: $NB(SelCond(D_1)$ Hor-Int $SelCond(D_2))$. The initial problem is formulated in the user ontology, and translated to the algorithm ontology and further to data source ontologies, by using the necessary mappings and conversion functions as described in Section 4.2. We have seen that in the case of Naive Bayes algorithm, the sufficient statistics are given by counts of tuples $(attribute, value, class)$, which determine a matrix of size $|A| \cdot |V| \cdot |C|$, where $|A|$ is the number of attributes of the data set used by the Naive Bayes algorithm (in this case, $D = SelCond(D_1)$ Hor-Int $SelCond(D_2)$, $|V|$ is

the maximum number of values that an attribute can take, and $|C|$ is the total number of classes. We also denote by $|D_1|$ and $|D_2|$ the number of instances in the data sets D_1 and D_2, respectively. Thus, in order to execute the workflow, the following query needs to be answered by the query answering engine:

counts(attribute, value, class)

FORALL (attribute, value, class)

FROM ($SelCond(D_1)$ Hor-Int $SelCond(D_2)$.

The query decomposition component decomposes this query into sub-queries $Q(D_1)$ and $Q(D_2)$ corresponding to the two data sources, as follows:

$counts(attribute, value, class)$	$counts(attribute, value, class)$
FORALL($attribute, value, class$)	FORALL($attribute, value, class$)
FROM($SelCond(D_1)$)	FROM($SelCond(D_2)$)

where $+_{matrix}$, representing the addition of two matrices of equal size, is used to compose the answers to these sub-queries.

The query optimizer takes as input the decomposed query and enumerates the possible plans for executing this query. There are four possible plans, which we show in Figure 5.10. For each plan, everything below the dashed horizontal line is executed at the remote data sources, and everything above is executed at the central place. The plans that do not satisfy the data sources constraints are eliminated. In our case, we assume that all four plans are possible.

We proceed to the calculation of the cost of each plan. We assume that the user wants to optimize only the communication cost, expressed as a function of $|D_1|, |D_2|, |A|, |V|, |C|$ defined above. We assume $|D_1| = 100$, $|D_2| = 50$, $|A| = 7$, $|V| = 4$, $|C| = 3$. Then,

- $cost(P_1) = |D_1|(|A| + 1) + |D_2|(|A| + 1) = (|A| + 1)(|D_1| + |D_2|) = 1200$,
- $cost(P_2 = |D_1|(|A| + 1) + |A||V||C|) = 884$,
- $cost(P_3) = |A||V||C| + |D_2|(|A| + 1) = 484$, and
- $cost(P_3) = 2 \cdot |A||V||C| = 168$.

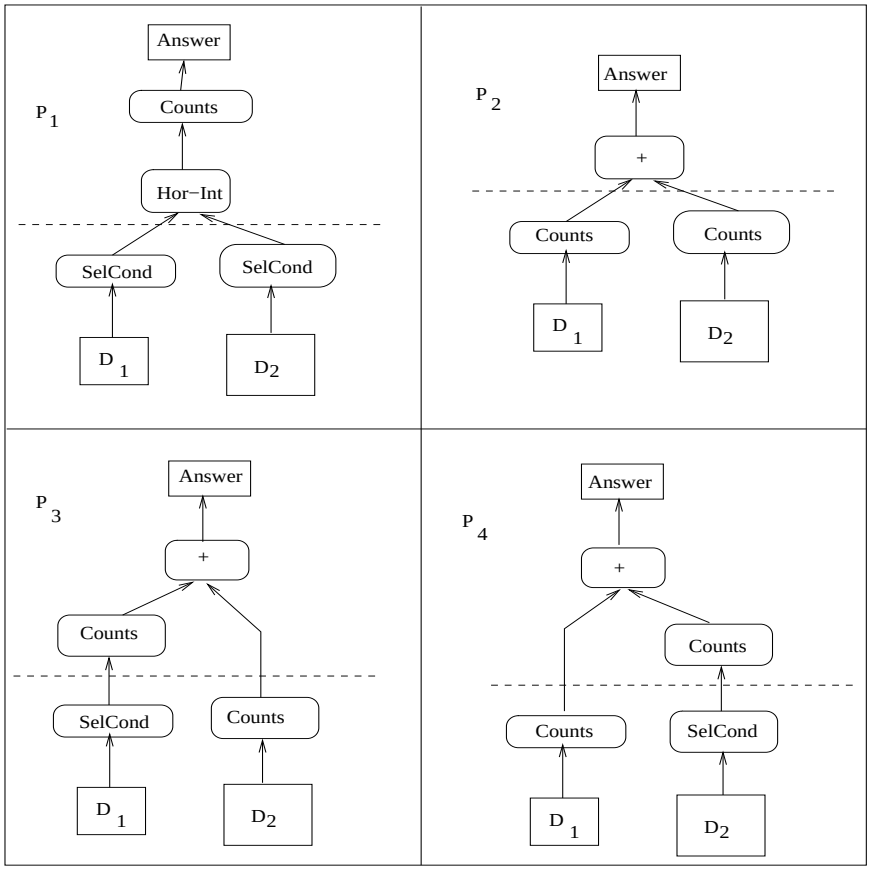

Figure 5.10 The four plans found by the query optimizer for Naive Bayes example. The operators below the dotted line are executed at the remote data sources, and the operators above the dotted line are executed at the central place

Thus, it turns out that the plan P_4 is the best and it will be sent further to the execution engine. The count matrices received back as answers are added up using the $+_{matrix}$ operation to get the final answer to the initial query.

5.6 Summary and Discussion

In this Chapter we have shown how a system for answering statistical queries from distributed heterogeneous autonomous data sources can be designed. Our approach draws on much of the existing literature on data integration and query optimization [Garcia-Molina *et al.*, 1997; Arens *et al.*, 1993; Knoblock *et al.*, 2001; Levy, 1998; Draper *et al.*, 2001] etc. Hence, it shares some of the features of existing data integration platforms. But it also includes some novel features.

Few of the existent systems take into account semantic relationships between values of attributes used to describe instances (e.g., taxonomies over attribute values) in individual data sources. The tools we introduced in Chapter 4 allow us to generalize various information integration approaches to work in settings where taxonomies over attributes are specified.

One important characteristic of our system consists of a clear separation between a user perspectives and the procedures used for query answering. This allows users to replace their ontologies *on the fly*, making it attractive for query answering tasks that arise in exploratory data analysis wherein scientists might want to experiment with alternative ontologies.

We proposed a query optimization algorithm similar to the algorithm in [Rodriguez-Martinez and Roussopoulos, 2000], where code can be shipped in order to minimize the amount of information transmitted over the network, if a data source constraints allows this. As opposed to the algorithm in [Rodriguez-Martinez and Roussopoulos, 2000] our algorithm works for general data sources (not only relational databases) and for the extended set of operators introduced in Section 3.4 (not only for relational operators). Furthermore, by defining ontologies associated with the data sources and mappings between ontologies, queries over semantically heterogeneous data sources can be answered.

In related work, Lambrecht and Kambhampati [1999] present a method for reducing the

amount of network traffic generated while executing an information gathering plan, by re-ordering the sequence in which queries are sent to remote information sources. Data source descriptions are used to assist in ordering the queries.

INDUS, a federated query-centric approach to learning classifiers from distributed heterogeneous autonomous data sources (described in the next Chaper) is implementing the query answering system designed in this Chapter.

6 INDUS: A FEDERATED QUERY-CENTRIC APPROACH TO LEARNING CLASSIFIERS FROM DISTRIBUTED HETEROGENEOUS AUTONOMOUS DATA SOURCES

6.1 Overview

Our work has contributed to the design and development of INDUS (INtelligent Data Understanding System) (see Figure 6.1). INDUS initially a federated, query-centric approach to information integration from distributed heterogeneous data sources [Reinoso-Castillo, 2002; Reinoso-Castillo *et al.*, 2003] has been substantially redesigned and extended to yield a system for learning classifiers from distributed heterogeneous autonomous data sources.

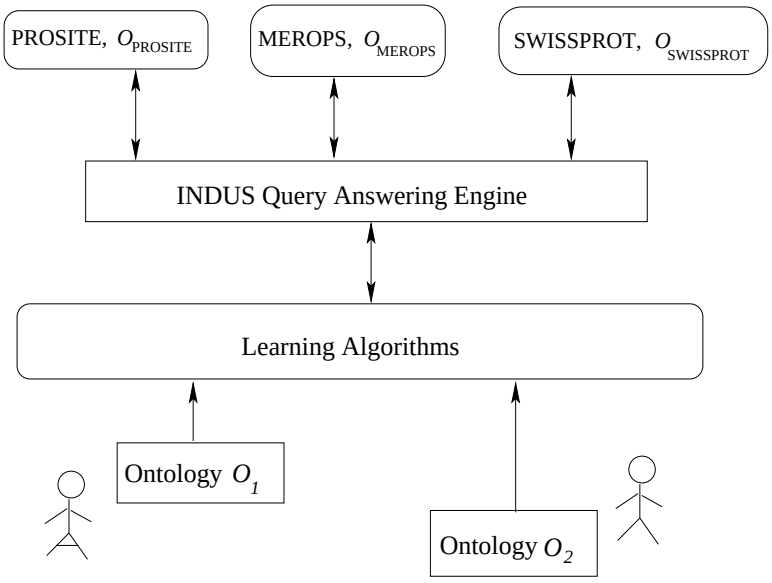

Figure 6.1 INDUS: Intelligent Data Understanding System. Three data sources are shown: $PROSITE$, $MEROPS$ and $SWISSPROT$ together with their associated ontologies. Ontologies O_1 and O_2 are two different user ontologies

The choice of the federated (as opposed to data warehouse) and query centric (as opposed

to source centric) approach to information integration was motivated by characteristics of a class of scientific applications of data-driven knowledge acquisition that are of interest to us. A detailed discussion of the design rationale of INDUS can be found in [Reinoso-Castillo *et al.*, 2003]. In brief, a federated approach lends itself much better to settings where it is desirable to postpone specification of the user ontology, O_U, and the mappings, $M(O_i, O_U)$, between data source specific ontologies, $O_1, \cdots, O_K$, and user ontology, O_U, until when the user is ready to use the system. The choice of a query centric approach in INDUS enables users the desired flexibility in querying data from multiple autonomous sources in ways that match their own context or application specific ontological commitments (whereas in a source centric approach, what the data from a source should mean to a user are determined by the source).

It is exactly the choice of federated and query centric approach that makes INDUS suitable for being transformed into a system for learning from distributed heterogeneous autonomous data sources. We have seen in the previous chapters that we can design algorithms for learning from distributed heterogeneous data by separating the learning task into information extraction and hypothesis generation components, which translates to formulating statistical queries, decomposing them into sub-queries corresponding to the data sources of interest, answering the sub-queries and composing the answers to these sub-queries into an answer to the initial query. Through the means of a query answering engine this process can be made transparent to the learning algorithm, and thus the learning algorithm becomes independent of the data given the sufficient statistics provided by the query answering engine.

As an information integration system, INDUS can be used to answer queries from distributed heterogeneous autonomous data sources. Thus, it is appropriate to transform the technology involved there into an INDUS query answering engine. By linking a set of learning algorithms to this engine, we obtain a learning system as the one designed in Chapter 5.

Weka [Witten and Frank, 1999] is a large and popular collection of machine learning algorithms (for classification, regression, clustering, association rules) and machine learning tools (for data pre-procession, visualization) implemented in Java. The algorithms and tools in Weka can either be applied directly to a dataset or called from other Java programs. They

could also be used as a starting point in developing new machine learning algorithms. However, the design of the algorithms in Weka does not take into account the separation between information extraction and hypothesis generation, and thus the resulting implementations are not independent of data, making it difficult to transform these algorithms into algorithms for learning from distributed heterogeneous autonomous data sources. Furthermore, most of the time the data is kept in the memory, which makes it impossible to run Weka algorithms on very large data sets.

In the next section, we show how a large class of Weka algorithms can be modified to enforce the separation of concerns between information extraction and hypothesis generation components of the algorithm. The resulting implementations provide a scalable and efficient approach to learning classifiers from large distributed semantically heterogeneous data sources.

We link the resulting algorithms to INDUS query answering engine and thus obtain a system for learning from distributed heterogeneous autonomous data sources.

6.2 From Weka to AirlDM to INDUS

AirlDM is a collection of machine learning algorithms, which are data source independent through the means of sufficient statistics and data source wrappers. They work with general data sources where data can be stored in any format as long as wrappers for accessing and getting sufficient statistics from those data sources are provided. Some of the algorithms in AirlDM are adapted from Weka implementations by separating the information extraction and hypothesis generation components.

Figure 6.2 shows the general architecture of AirlDM. As can be seen a learning algorithm is regarded as a `TRAINER` that generates a `HYPOTHESIS` from `SUFFICIENT STATISTICS`. Each `DATA SOURCE` is wrapped by a `DATA SOURCE WRAPPER`. The `TRAINER` `registers sufficient statistics` with the `DATA SOURCE WRAPPER` which `populates` them by accessing the corresponding `DATA SOURCE`. Once the `SUFFICIENT STATISTICS` are populated, they are used by the `TRAINER` to `get parameters` that are needed to build a current `HYPOTHESIS`. This process may repeat a few time (e.g., for decision tree algorithm). When a hypothesis is built, a `USER`

can `get hypothesis` from the `TRAINER` and use it to classify new unseen data.

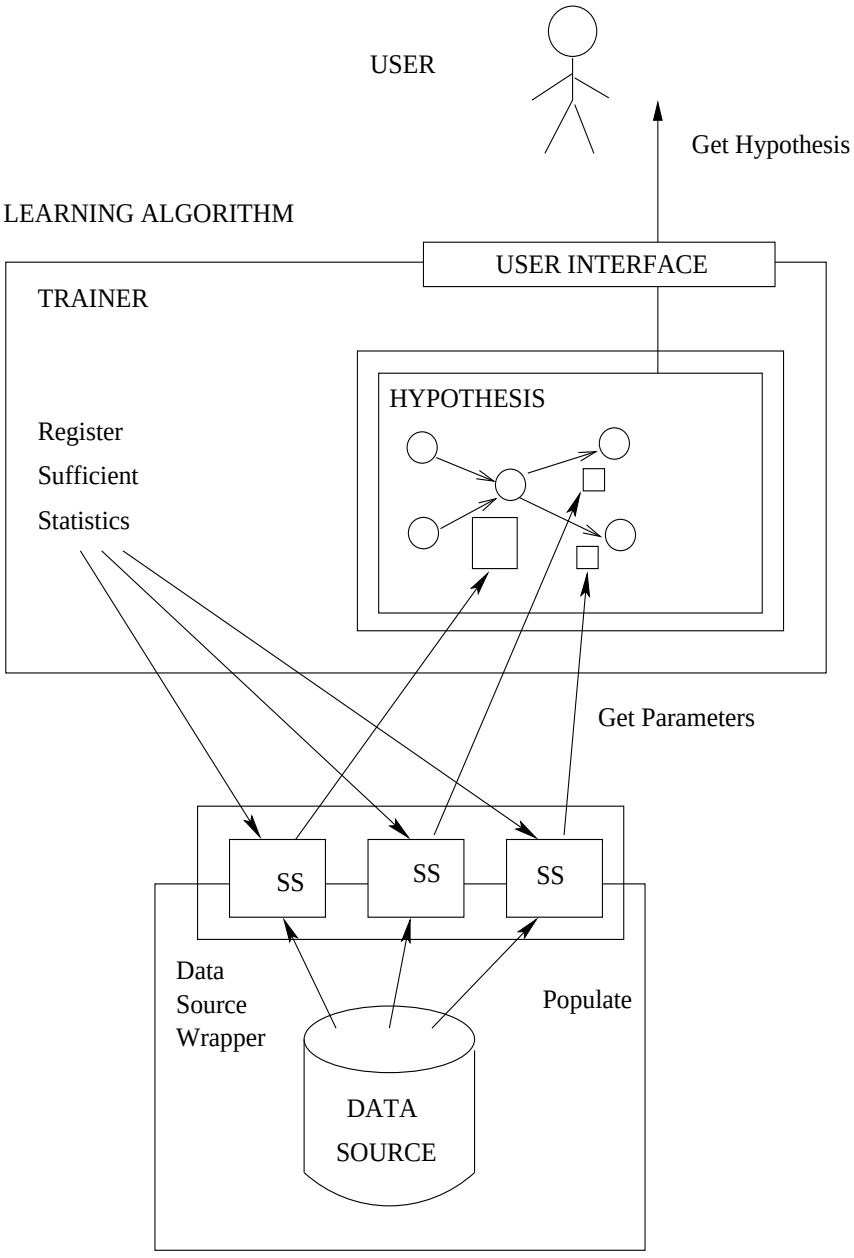

Figure 6.2 AirlDM: Data source independent learning algorithms through the means of sufficient statistics and wrappers

AirlDM is an open source software issued under the GNU General Public License. In the current release, we provide wrappers for data that can be seen as a single table (INDUS wrapper, Weka wrapper), as a collection of tables (multi relational data wrapper) or as a sequence (sequence wrapper). We have implemented sufficient statistics of type joint counts, which are the sufficient statistics needed by a large class of algorithms (e.g., Naive Bayes

[Mitchell, 1997], Bayes Networks [Pearl, 2000; Jensen, 2001], Bags of Words [Mitchell, 1997], Relational Learning [Friedman *et al.*, 1999; Atramentov *et al.*, 2003], NB-k [Silvescu *et al.*, 2004a], Decision Trees with a variety of splitting criteria [Buja and Lee, 2001] etc.). The algorithms currently implemented are Naive Bayes and Decision Tree Algorithm.

Because of the modular design of AirlDM and the clear separation of concerns between hypothesis generation and information extraction, AirlDM can be easily linked to INDUS to obtain a system for learning from heterogeneous distributed autonomous data sources. Thus, we have written an INDUS wrapper that provides sufficient statistics to the trainer and linked it with the AirlDM implementations of Naive Bayes and Decision Tree algorithms. Using that, we have implemented algorithms for learning Naive Bayes and Decision Tree classifiers from horizontally and vertically distributed data sources by having the query answering engine register the sufficient statistics that it gets from the trainer with the corresponding wrappers and composing the statistics populated by the wrappers into the sufficient statistics needed by the trainer. Thus, in the case of horizontally distributed data, each count statistic is registered with the wrapper of each distributed data source and the answers are added up to get the overall count. In the case of vertically fragmented data, the query answering engine identifies the wrapper that can be used to populate each sufficient statistic count and the answer is sent back to the trainer.

Therefore, we can achieve learning from distributed data in a way which is transparent to the learning algorithm, meaning that from the algorithm point of view it makes no difference if the data comes from a single or multiple data sources or if these data sources are represented as relational tables or flat file or any other format. Furthermore, if the distributed data sources are heterogeneous, the query answering engine can perform mappings from data sources ontologies to user ontology and the algorithms remain unchanged.

6.3 Case Study

6.3.1 Data Sources

To the best of our knowledge, there are no widely used benchmark data sets for evaluation of systems for learning classifiers from semantically heterogeneous distributed data sources, therefore we need to develop benchmark data sets. One appropriate data for the algorithms that we design might be Census Data (www.thedataweb.org). An online data library made available by the US Census Bureau, Bureau of Labor Statistics, and the Centers for Disease Control along with an access tool (DataFerrett - Federated Electronic Research, Review, Extraction, and Tabulation Tool) makes it easier to collect census data.

Using these tools Ronny Kohavi and Barry Becker (Data Mining and Visualization Silicon Graphics) [Kohavi, 1996] extracted an income census data set UCI/ADULT from the 1994 census bureau database found at http://www.census.gov/ftp/pub/DES/www/
welcome.html and donated it to the UCI Machine Learning Repository [Blake and Merz, 1998]. The classification task is to determine if the salary of a person represented by a record is greater or less than $50,000. The initial data was split into train/test in approximately 2/3, 1/3 proportions. There are 48842 instances (train=32561, test=16281). Each instance is described by 5 continuous attributes (age, education-num, capital-gain, capital-loss, hours-per-week) and 8 nominal attributes (workclass, education, marital-status,occupation, relationship, race, sex, native-country). There are 7% missing values. The class distribution is as follows: probability for the label $> 50,000$ is 23.93%, probability for the label $<= 50,000$ is 76.07%.

Terran Lane and Ronny Kohavi (Data Mining and Visualization Silicon Graphics) extracted another income census data set UCI/CENSUS-INCOME from the 1994 census bureau database and donated it to the UCI Machine Learning Repository. The initial data extracted was also split into train/test in approximately 2/3, 1/3 proportions. There are 199523 instances in the training set and 99762 instances in the test set, and 40 attributes (7 continuous, 33 nominal), which makes this data set much bigger than the original UCI/ADULT data set (approximately 100Mb compared to 5Mb). The classification task is to determine the income level for the person represented by a record. Incomes have been binned at the $50,000 level

to present a binary classification problem, much like the original UCI/ADULT database.

We used the census data UCI/ADULT and UCI/CENSUS-INCOME to test our algorithms for learning Naive Bayes classifiers from distributed, semantically heterogeneous data. We tested the algorithms for learning Naive Bayes classifiers from horizontally distributed data and from vertically distributed data with UCI/CENSUS-INCOME. As we do not have an ontology editor to create mappings between ontologies, we used UCI/ADULT data (for which we hand-crafted mappings) to test the algorithms for learning Naive Bayes classifiers from semantically heterogeneous horizontally/vertically distributed data. Before using these data sets for learning Naive Bayes classifiers, we filled in missing values and discretized the continuous attributes using Weka.

Running Weka implementation of Naive Bayes algorithm on UCI/CENSUS-INCOME ends with "Out of Memory" error. However, AirlDM with INDUS wrapper gives an accuracy of 76.2174%. The accuracy on UCI/ADULT is 84.2516%. We evaluate our algorithms for learning from heterogeneous distributed data by comparison with the batch algorithms whose accuracy is shown here. We expect to obtain the same accuracy in the case of the algorithms for learning from distributed data, as these algorithms are provably exact, but probably different results for the algorithms for learning from heterogeneous data, as associating ontologies may increase or decrease the accuracy [Zhang and Honavar, 2003].

6.3.2 Learning NB Classifiers from Distributed Data

As described above, UCI/CENSUS-INCOME data consists of a training (2/3) and a test (1/3) set obtained by splitting the original data into two subsets. To generate two horizontally fragmented distributed data sets, we randomly split the training set further into two subsets D_1^h, D_2^h of approximately same size (1/2,1/2). To generate two vertically fragmented distributed data sets, we randomly split the attribute set into two attribute subsets of approximately the same size (1/2,1/2). The data corresponding to the first attribute subset goes into D_1^v and the data corresponding to the second attribute subset goes into D_2^v. We apply the algorithm for learning Naive Bayes classifiers from horizontally (vertically) distributed data sources on the subsets D_1^h, D_2^h (and D_1^v, D_2^v, respectively). The results are shown in Table 6.1.

They prove that indeed the algorithms for learning from distributed data that we designed are exact with respect to their batch counterparts, as we get the same results in all three cases.

Table 6.1 Learning from distributed UCI/CENSUS-INCOME data sources

Distribution Type	*Accuracy %*	*Error %*	*Correct*	*Incorrect*
Horizontal	76.2174	23.7826	855481	23726
Vertical	76.2174	23.7826	855481	23726
Centralized	76.2174	23.7826	855481	23726

6.3.3 Learning NB Classifiers from Heterogeneous Distributed Data

As described above, UCI/ADULT data consists of a training (2/3) and a test (1/3) set obtained by splitting the original data into two subsets. We use the same procedure as in Section 6.3.2 to further split the training data into two horizontal subsets D_1^h, D_2^h of approximately same size (1/2,1/2) and then into two vertical subsets D_1^v, D_2^v of approximately same size (1/2,1/2).

We used a software provided by Kang *et al.* [2004] to generate AVT's over the data sets D_{test}, D_1^h, D_2^h, D_1^v, D_2^v, respectively. The taxonomies for the attribute *Ocupation* are shown for the user (test) data set, data set D_1^h and data set D_2^h in Figures 6.3, 6.4 and 6.5, respectively. Graphviz, an open source graph drawing software from AT&T Labs Research [Gansner and North, 2000], was used to draw these figures.

We hand-crafted mappings between the taxonomies associated with D_1^h, D_2^h, D_1^v, D_2^v, respectively, to the taxonomy associated with D_{test} and chose a user level of abstraction such that some attributes in the distributed data sources are under-specified, while others are over-specified. To be able to deal with the under-specified values, we made the assumption that all the data come from the same distribution and we used a distribution inferred from the user data to fill in the under-specified values.

We apply the algorithm for learning Naive Bayes classifiers from horizontally (vertically) distributed heterogeneous data sources to the subsets D_1^h, D_2^h (and D_1^v, D_2^v, respectively). The results are shown in Table 6.2 and they confirm our theoretical results, as the same accuracy

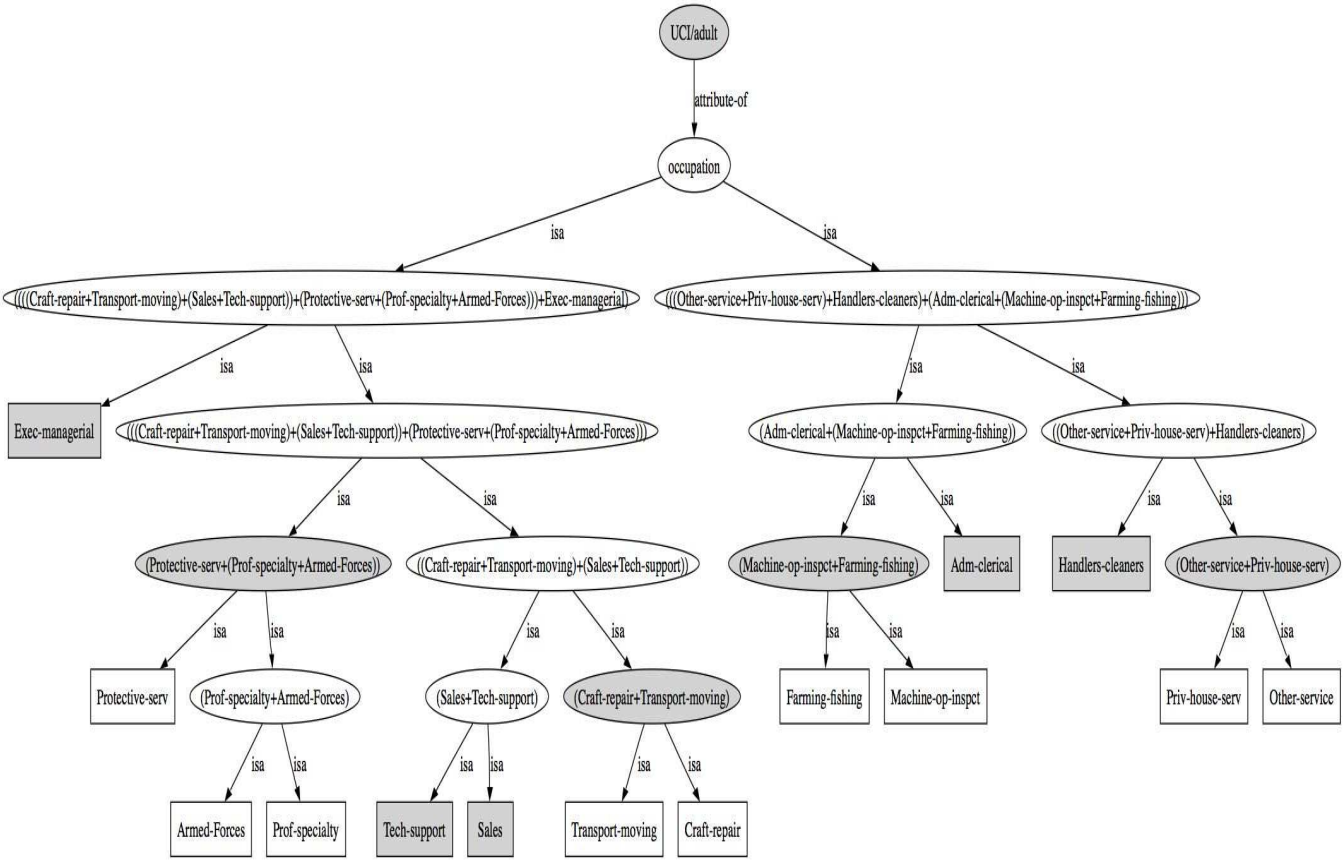

Figure 6.3 Taxonomy for the attribute *Ocupation* in user (test) data. The filled nodes represent the level of abstraction specified by the user

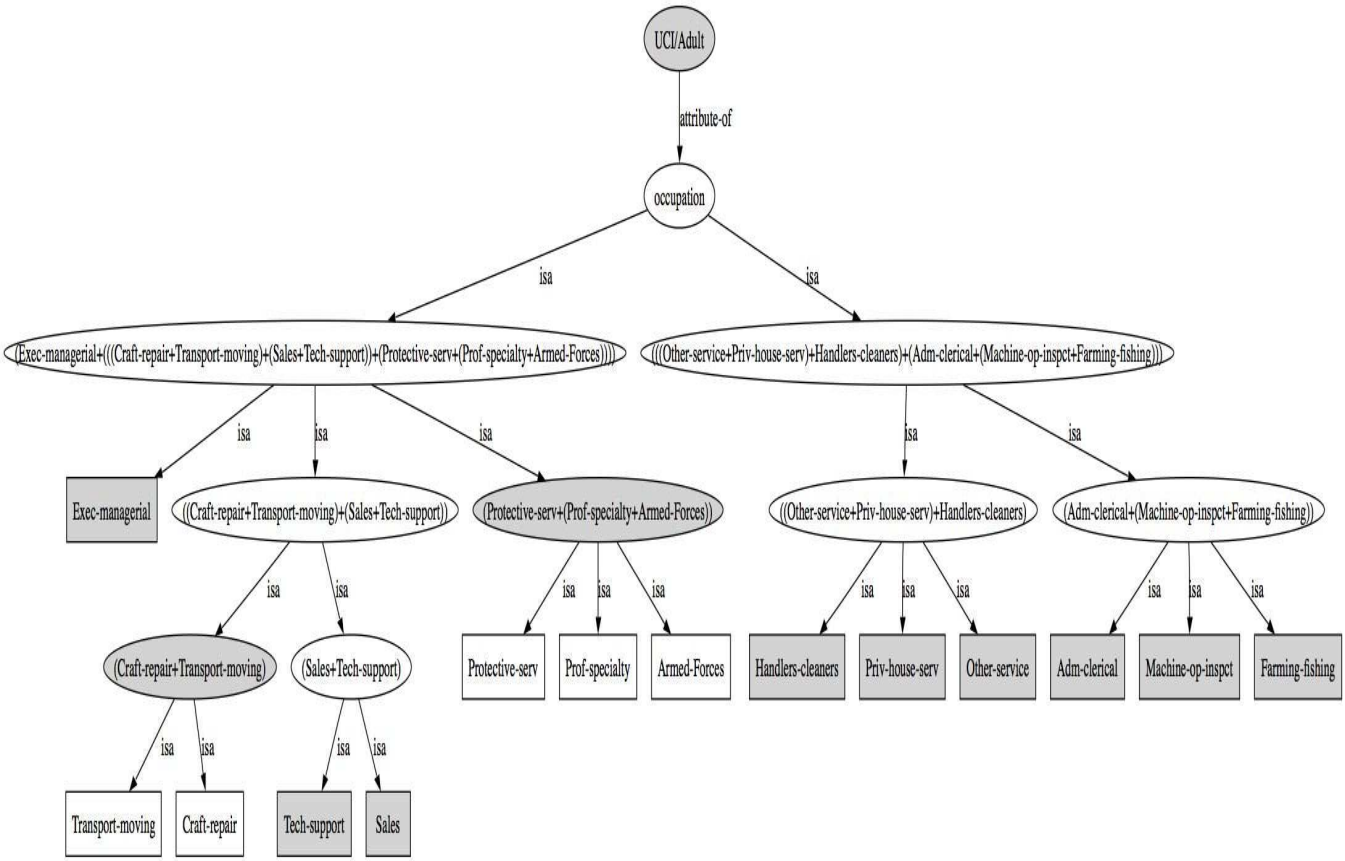

Figure 6.4 Taxonomy for the attribute *Ocupation* in the data set D_1^h. The filled nodes represent the level of abstraction determined by the user cut. Values *Priv-house-serv, Other-service, Machine-op-inspct, Farming-fishing* are over specified with respect to the user cut

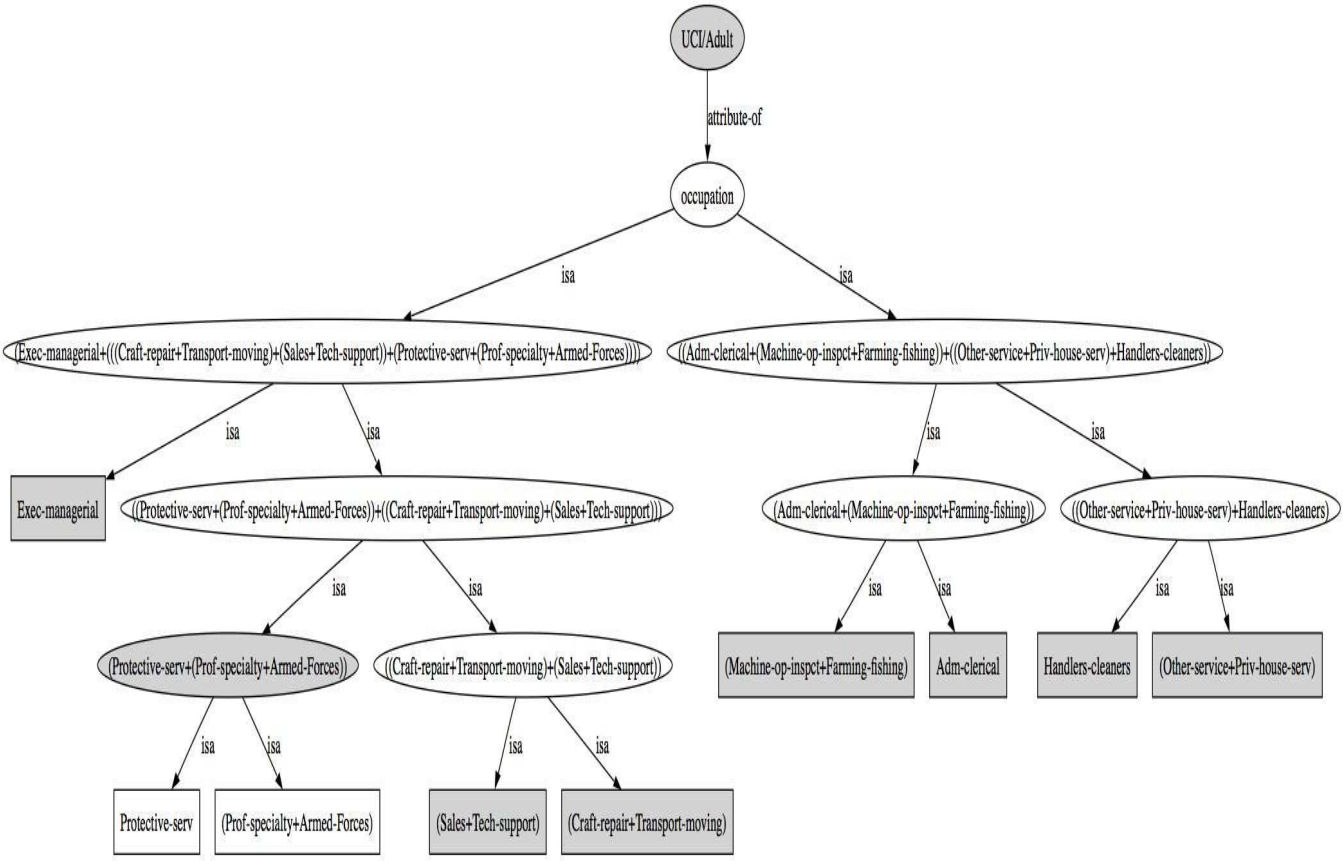

Figure 6.5 Taxonomy for the attribute *Ocupation* in the data set D_2^h. The filled nodes represent the level of abstraction determined by the user cut. The value *(Sales+Tech-support)* is underspecified with respect to the user cut

is obtained in the case of centralized, horizontal and vertical data distributions.

Table 6.2 Learning from heterogeneous UCI/ADULT data sources

Distribution Type	*Accuracy%*	*Error%*	*Correct*	*Incorrect*
Horizontal	83.6435	16.3565	13618	2663
Vertical	83.6435	16.3565	13618	2663
Centralized	83.6435	16.3565	13618	2663

6.4 Summary and Discussion

This Chapter contains the overview of a system for learning from distributed heterogeneous autonomous data sources to which the work in this dissertation contributed. We show how a large class of algorithms in Weka as well as new learning algorithms, such as [Silvescu *et al.*, 2004a; Atramentov *et al.*, 2003] etc. can be implemented in AirlDM, a publicly available software which is designed in terms of the separation between information extraction and hypothesis generation components of a learning algorithm, and how AirlDM can be used further with INDUS query answering engine to design algorithms for learning from heterogeneous distributed data. A case study using census data is also presented.

The development of ontology servers has been an active research area over recent years and a number of systems have been designed and implemented [Farquhar *et al.*, 1996; Papageorgiou *et al.*, 2003; Bernstein *et al.*, 2000; Bernstein, 2003]. Typically such systems store ontological information in a knowledge-base that works in tandem with a database system to produce a unified view of heterogeneous distributed data. Current research issues include the management of changing ontologies in a distributed environment [Heflin *et al.*, 1999], support for dynamic and multiple ontologies [Heflin and Hendler, 2000], and resusable ontologies [Musen, 1998].

We used a software provided by Kang *et al.* [2004] to generate AVT's over the set of attributes in our domain. However, we defined the mappings between ontologies manually. Bao and Honavar [2004] present P-OWL (Package-based OWL), an extension of OWL, a widely used ontology language that supports modular design, adaptation, use and reuse of

ontologies. P-OWL localizes the semantics of entities and relationships in OWL to modules called packages. Ontomill, a collaborative ontology tool that includes an ontology editor and a reasoner, is also described. P-OWL and the associated tool will greately facilitate collaborative ontology construction, use and reuse, as well as mapping definition.

In the future, we plan to perform the evaluation of the proposed algorithms on a broad range of distributed semantically heterogeneous data from a number of domains including bioinformatics [Yan *et al.*, 2004a; 2004b; Andorf *et al.*, 2004] and security information [Kang *et al.*, 2004a], among others, along a number of dimensions including in particular, characteristics of data sources (structure of data sources, query and processing capabilities, complexity of associated ontologies and mappings between ontologies, size of the data sets, prevalence of partially missing attribute values as a consequence of integration of data described at multiple levels of granularity), characteristics of algorithms (e.g., types of statistics needed for learning), and performance criteria (quality of results produced relative to the centralized counterparts, computational resource, bandwidth, and storage usage).

7 CONCLUSIONS

7.1 Summary

Efficient learning algorithms with provable performance guarantees for learning from distributed heterogeneous data constitute a key element of any practical approach to data driven discovery and decision making using large, autonomous data repositories that are becoming available in many domains (e.g., biological sciences, atmospheric sciences).

In this dissertation, we have precisely formulated the problem of learning from distributed data sources and described a general strategy for transforming standard machine learning algorithms that assume centralized access to data in a single location into algorithms for learning from distributed data. This strategy relies on the separation of a learning algorithm into an information extraction component that gathers sufficient statistics needed for learning and a hypothesis generation component that uses these sufficient statistics to generate a current hypothesis.

We have demonstrated the application of this strategy to devise several algorithms (Naive Bayes, Decision Trees, Perceptron, Support Vector Machines and k-NN) for induction of classifiers from distributed data. The resulting algorithms are *provably exact* in that the hypothesis constructed from distributed data is identical to that obtained by the corresponding algorithm when it is used in the centralized setting. This ensures that the entire body of theoretical (e.g., sample complexity, error bounds) and empirical results obtained in the centralized setting carry over to the distributed setting.

We have introduced a statistical query language consisting of operators for formulating and manipulating statistical queries and showed how the algorithms for learning from distributed data can be further extended to algorithms for learning from semantically heterogeneous

distributed data by extending data sources and operators with ontologies in a way that ensures sound and complete answers to statistical queries in the presence of ontologies. Learning from such data sources reduces to learning from partially specified data. To show how this works, we have designed algorithms for inducing classifiers (Naive Bayes, Decision Trees, Threshold Functions, SVMs, k-NNs) from semantically heterogeneous distributed data.

As gathering sufficient statistics from heterogeneous distributed data under various constraints imposed by data sources turns out to be very important for solving the problem of learning from distributed data, we have designed a query answering engine that receives queries from learning algorithms, decomposes them into sub-queries according to the distributed data sources, finds an optimal plan for executing sub-queries, executes the plan and composes the individual answers it gets from the distributed data sources into an answer to the initial query.

The algorithms and strategies designed through this dissertation are implemented in AirlDM and INDUS and a case study proving how they work is presented.

7.2 Contributions

The major contributions of this dissertation include:

- **A General Strategy for Design of Algorithms for Learning Classifiers from Distributed Data** [Caragea *et al.*, 2004d]

  We have proposed a general strategy for design of algorithms for learning classifiers from distributed data based on a separation of concerns between hypothesis generation and information extraction (statistical query answering) [Caragea *et al.*, 2004d]. We have designed algorithms with strong performance guarantees (relative to their centralized counterparts) for learning decision tree [Caragea *et al.*, 2003], support vector machine [Caragea *et al.*, 2001], nearest neighbor, perceptron and naive Bayes classifiers from distributed data.

- **A General Framework for Design of Algorithms for Learning Classifiers from Semantically Heterogeneous Data** [Caragea *et al.*, 2004b].

We have proposed a framework for design of algorithms for learning from semantically heterogeneous data based on the extension of data sources and operators with ontologies. We have showed how we can answer queries from semantically heterogeneous data using this framework and designed algorithms for learning from such data based on approaches for learning from partially specified data.

- **Design of INDUS Query Answering Engine** [Caragea *et al.*, 2004a]

  We have showed how we can transform the INDUS information integration system into an INDUS query answering engine that can answer statistical queries from semantically heterogeneous data sources under a variety of constraints and assumptions motivated by application scenarios encountered in practice.

- **An Open Source Package Containing Data Source Independent Machine Learning Algorithms** [Silvescu *et al.*, 2004b]

  As the design of the algorithms for learning from distributed heterogeneous data sources relies on the decomposition of learning into statistics gathering and hypothesis generation, we materialized this decomposition in AirlDM, a data source independent collection of learning algorithms through the means of sufficient statistics and data source wrappers. AirlDM contains an INDUS wrapper that can be used to answer queries from semantically distributed data sources and thus, we obtain implementation of algorithms for learning from semantically distributed data sources.

7.3 Future Work

Several future research directions are outlined below:

- **Evaluation of the Query Optimization Algorithm in INDUS**

  Design of algorithms for learning from distributed data described in this paper has been motivated by the desirability of performing as much of the processing of data as feasible at the sites where the data and computing resources are available to avoid retrieving

large volumes of data from remote sites. The applicability of the proposed approach in practice depends on whether information requirements of the learning algorithm L under consideration can be met under the constraints imposed by the distributed setting and the time, memory, and communication costs of the resulting algorithm relative to the other alternatives (e.g., gathering all of the data in a centralized site and then applying the centralized learning algorithm if such a solution is allowed by the constraints Z). It is of interest to implement the techniques described in Chapter 5 for query optimization ontology editor extension to more general ontologies in INDUS query answering engine and experiment with different choices of constraints Z (e.g., privacy constraints in knowledge acquisition from clinical records) that arise in practice.

- **System Evaluation**

  In the future, we plan to perform the evaluation of the proposed algorithms along a number of dimensions including in particular, characteristics of data sources (structure of data sources, query and processing capabilities, complexity of associated ontologies and mappings between ontologies, size of the data sets, prevalence of partially missing attribute values as a consequence of integration of data described at multiple levels of granularity), characteristics of algorithms (e.g., types of statistics needed for learning), and performance criteria (quality of results produced relative to the centralized counterparts, computational resource, bandwidth, and storage usage).

- **Design of Approximate/Cumulative/Incremental Learning Algorithms**

  This dissertation has focused primarily on algorithms for learning from distributed data that are provably exact relative to their centralized counterparts. In many applications, it would be of interest to relax the exactness requirement leading to provably approximate algorithms (based on resource constrained approximations of sufficient statistics). Also of interest are extensions of the proposed approach to *cumulative* and *incremental* learning scenarios [Caragea *et al.*, 2001; Polikar *et al.*, 2001; Caragea *et al.*, 2004c].

- **Learning from Multi-Relational Tables**

  In related work, [Atramentov *et al.*, 2003] have developed algorithms for learning from multiple tables in a relational database. It is of interest to explore approaches similar to those described in this dissertation for learning from distributed relational databases as well as heterogeneous distributed data which are presented by INDUS as if they were set of relations.

- **Multi Agent Data Mining**

  Bromberg *et al.* [2004] , have defined the problem of multi-agent data mining, which is an extension to the framework for learning from distributed data. In multi-agent data mining, the agents have limited resources and are self-interested, but they can achieve their goals by communicating and exchanging information with other self-interested agents. Thus, mechanisms for knowledge production and coordination, similar to those in economics, need to be developed. We assume that there are hundreds of agents in such a framework, so one agent cannot communicate with all the agents in the system but just with a small subset of agents. One natural extension to the framework in [Bromberg *et al.*, 2004] is to associate ontologies with each agent in the system. Here, we do not have an overall integration ontology, but we can define integration ontology for the neighborhood of an agent.

- **Semantically Heterogeneous Distributed Data Visualization**

  Zhang *et al.* [2003] describe Limn Matrix, a system that interactively display density plots for large, distributed data. This system makes use of a novel hierarchical indexing technique that dramatically reduces the delay through the network. The framework introduced in Chapter 4 in this dissertation could be used to extend Limn Matrix to visualize semantically heterogeneous data.

- **Ontology Language/Learning/Manipulation**

  Recent development of the Semantic Web [Berners-Lee *et al.*, 2001] calls for large-scale and well-maintained ontologies. However, little attention has been paid to the formalism

of building large-scale ontologies in distributed environments where both the ontology integration and the ontology independence are important. In related work, Bao and Honavar [2004] have proposed an extended ontology language to support modularity and locality in semantics. In future work, we plan to use this language to design an ontology editor that helps in building ontologies over distributed data sources and also allows to define mappings (interoperation constraints) between different ontologies.

- **Ontology-Extended Workflows**

  Pathak *et al.* [2004] have developed ontology-extended workflow components and semantically consistent methods for assembling such components into complex ontology-extended component-based workflows. The result is a sound theoretical framework for assembly of semantically well-formed workflows from semantically heterogeneous components. In this case, there is no integration hierarchy for all the ontologies associated with components in the workflow, as some of them may be unrelated. Instead an integration ontology is found for every set of ontologies corresponding to neighboring (source, target) components. Work in progress is aimed at design and implementation of an environment for workflow assembly and execution from semantically heterogeneous software components, ontologies and user supplied mappings between ontologies.

- **Applications**

  Some of the work in progress is aimed at application of the proposed algorithms to knowledge acquisition tasks that arise in applications in computational biology [Yan *et al.*, 2004a; 2004b; Andorf *et al.*, 2004], information security [Kang *et al.*, 2004a], and related domains.

GLOSSARY

aggregate operators operators used to compute aggregate statistics over some data

answer shipping when the query is executed at the data site (via shipping the code if it is not already available) and only the results of the query are shipped at the client site

attribute feature

attribute value taxonomy *is-a* hierarchy over a set of attributes

attribute values values that an attribute can take

autonomous data sources data sources that are not controlled by the learner and may impose constraints on the learner

background knowledge the information that the learner has about the task before the learning process (e.g., simple answers are preferable over complex answers)

central resource repository where all the resources in the system are registered

central site where the information extraction is done

class label value of the class attribute

classification phase of an algorithm when new unlabeled examples are classified

classification task a task for which the learner is given experience in the form of labeled examples, and it is supposed to learn to classify new unlabeled examples

classifier the result of a learning algorithm

clients learning algorithms or users

completely specified instances instances for which all attributes are specified at the right level of abstraction

compositional operators operators that can be used to combine statistics extracted from several data sources

consistent hypothesis a hypothesis which is consistent with a set of labeled examples

consistent learner if it outputs a hypothesis which is consistent with a set of labeled examples

constraints a set of constraints imposed on the learner in a distributed setting coming from privacy concerns, storage issues, operations allowed, etc.

data access modules iterators

data operators operators whose inputs and outputs are data sets

data set a collection of examples; we assume that the examples are randomly chosen from an unknown distribution

data shipping when the data is shipped at the client site and the query is executed there

data-inflating operators whose volume is larger than the volume of the data that they are applied to

data-reducing operators return answers whose volume is smaller than the volume of the data that they are applied to

distributed data a collection of data sets distributed into a network

eager learning algorithm learning algorithms that do most of the work during learning phase

exact learning from distributed data an algorithm for learning from distributed data which outputs a hypothesis identical to the hypothesis output by its centralized counterpart

examples the learner is presented with labeled examples about a particular task

features an example is described by a set of features

filters data-reducing operators

heterogeneous data a collection of data sources that are heterogeneous in structure (e.g., flat file, relational database) or in content (different ontological commitments)

hierarchy categorization of a set according to an order

horizontal fragmentation type of data fragmentation wherein subsets of data tuples are stored at different sites

hypothesis the output of a learning algorithm

hypothesis generation component of a learning algorithm when the hypothesis is generated

information extraction component of a learning algorithm when the information needed for learning is gathered

instances examples

integrable hierarchies hierarchies that can be integrated according to an integration hierarchy

integrable ontologies a set of ontologies that can be integrated

integration hierarchy a hierarchy which can integrate a set of hierarchies (there exists partial injective mapping from the integrated hierarchies to the integration hierarchy)

integration ontology an ontology that can be used to integrate a set of ontologies

interoperation constraints constraints that need to be specified by the mappings between two ontologies

labeled example an example for which the class is not specified

labeled example an example for which the class is specified

lazy learning algorithm learning algorithms that do most of the work during classification phase

learner an algorithm or a computer program that is able to use the experience to improve its performance at some task

learning algorithm learner

learning from data learning from a set of training examples

learning phase of an algorithm when the hypothesis is generated

level of abstraction a cut through an *is-a* hierarchy

machine learning multidisciplinary field that brings together scientists from artificial intelligence, probability and statistics, computational complexity, information theory, etc.

message any unit of information sent over the network

model hypothesis

ontology a specification over objects, categories, properties and relationships used to conceptualize some domain of interest; a set of hierarchies

ontology mappings mappings between terms in two different ontologies

ontology-extended data sources data sources that have an ontology associated with them

operators operations that can be executed on a data set

over specified value value which is under the level of abstraction in an attribute value taxonomy

partial specified instances instances for which some attributes are not specified at the right level of abstraction

performance criteria measure the quality of the learning output in terms of accuracy, simplicity, efficiency, etc.

queries queries that the learner can pose in the process of learning

query answering engine where the queries are decomposed into sub-queries and the answers to sub-queries are composed into answers to the initial queries

selection condition a condition that specifies the data from which some statistics are gathered

servers data sources

statistic any function of data

statistical operators operators that output statistics about data

statistical query a query that returns a statistic

statistical query language set of operators used to formulate and manipulate statistical queries

sufficient statistic for a parameter a statistic that provides all the information needed to estimate the parameter

sufficient statistics for learning a hypothesis a statistic that provides all the information needed to learn a hypothesis

task a description of the task that the learner is trying to accomplish (e.g. a concept, a function, a language, etc.)

test set a set of unlabeled examples

training set a set of labeled examples

under specified value value which is above the level of abstraction in an attribute value taxonomy

user perspective given by an ontology and a set of interoperation constraints between this ontology and other ontologies in the system

vertical fragmentation type of data fragmentation wherein sub-tuples of data tuples are stored at different sites

wrappers used to access and retrieve information from data sources (iterators)

INDEX

BIBLIOGRAPHY

[Agrawal and Shafer, 1996] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8:962–969, 1996.

[Agrawal and Srikant, 2000] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 439–450, Dallas, Texas, May 2000.

[Amado *et al.*, 2003] N. Amado, J. Gama, and F. Silva. Exploiting Parallelism in Decision Tree Induction. In *Parallel and Distributed Computing for Machine Learning. In conjunction with the 14th European Conference on Machine Learning (ECML'03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, Cavtat-Dubrovnik, Croatia, September 2003.

[AMIAS, 2002] American Medical Informatics Association. *American Medical Informatics Association Symposium on Ontologies, Terminologies, and the Semantic Web for Clinical and Bio Scientists*, San Antonio, TX, November 2002.

[Andorf *et al.*, 2004] C. Andorf, D. Dobbs, and V. Honavar. Discovering protein function classification rules from reduced alphabet representations of protein sequences. *Information Sciences*, 2004. In press.

[Andrade *et al.*, 2003] H. Andrade, T. Kurc, J. Saltz, and A. Sussman. Decision tree construction for data mining on clusters of shared memory multiprocessors. In *Proceedings of the Sixth International Workshop on High Performance Data Mining: Pervasive and Data Stream Mining (HPDM:PDS'03). In conjunction with Third International SIAM Conference on Data Mining*, San Francisco, CA, May 2003.

[Arens *et al.*, 1993] Y. Arens, C. Chin, C. Hsu, and C. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[Aronis *et al.*, 1996] J. Aronis, V. Kolluri, F. Provost, and B. Buchanan. The WoRLD: knowledge discovery from multiple distributed databases. Technical Report ISL-96-6, Intelligent Systems Laboratory, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, 1996.

[Ashburner *et al.*, 2000] M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, M.A. Harris, D.P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J.C. Matese, J.E. Richardson, M. Ringwald, G.M. Rubin, and G. Sherlock. Gene ontology: tool for unification of biology. *Nature Genetics*, 25(1):25–29, 2000.

[Ashrafi *et al.*, 2002] M. Z. Ashrafi, D. Taniar, and K. A. Smith. A data mining architecture for distributed environments. In *Proceedings of the Second International Workshop on Innovative Internet Computing Systems*, pages 27–38, Kühlungsborn, Germany, June 2002.

[Atramentov *et al.*, 2003] A. Atramentov, H. Leiva, and V. Honavar. Learning decision trees from multi-relational data. In T. Horvth and A. Yamamoto, editors, *Proceedings of the 13th International Conference on Inductive Logic Programming*, volume 2835 of *Lecture Notes in Artificial Intelligence*, pages 38–56. Springer-Verlag, 2003.

[Bala *et al.*, 2002] J. Bala, S. Baik, A. Hadjarian, B. K. Gogia, and C. Manthorne. Application of a distributed data mining approach to network intrusion detection. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1419–1420, Bologna, Italy, 2002. ACM Press.

[Bao and Honavar, 2004] J. Bao and V. Honavar. Ontology language extensions to support semantics modularity and locality. In preparation, 2004.

[Barsalou and Gangopadhyay, 1992] T. Barsalou and D. Gangopadhyay. M(dm): An open framework for interoperation of multimodel multidatabase systems. *IEEE Data Engineering*, pages 218–227, 1992.

[Berners-Lee *et al.*, 2001] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[Bernstein *et al.*, 2000] P. Bernstein, A. Halevy, and R. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4), 2000.

[Bernstein, 2003] P.A. Bernstein. Applying model management to classical meta data problems. In *Proceedings of the Second Conference on Innovative Data Systems Research*, pages 209–220, Asilomar, CA, 2003.

[Bhatnagar and Srinivasan, 1997] R. Bhatnagar and S. Srinivasan. Pattern discovery in distributed databases. In *Proceedings of the Fourteenth AAAI Conference*, pages 503–508, Providence, RI, 1997. AAAI Press/The MIT Press.

[Blake and Merz, 1998] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. See http://www.ics.uci.edu/∼mlearn/MLRepository.html (retrieved 19 July 2004).

[Blockeel and Raedt, 1997] H. Blockeel and L. De Raedt. Relational knowledge discovery in database. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 199–212, 1997.

[Bonatti *et al.*, 2003] P. Bonatti, Y. Deng, and V. Subrahmanian. An ontology-extended relational algebra. In *Proceedings of the IEEE Conference on Information Integration and Reuse*, pages 192–199. IEEE Press, 2003.

[Bonnet *et al.*, 2001] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14. Springer-Verlag, 2001.

[Bradley and Mangasarian, 2000] P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13(1):1–10, 2000.

[Breiman *et al.*, 1984] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and regression trees*. Wadsworth, Monterey, CA, 1984.

[Bright *et al.*, 1992] M.W. Bright, A.R. Hurson, and S.H. Pakzad. A taxonomy and current issues in multibatabase systems. *Computer Journal*, 25(3):5–60, 1992.

[Bromberg *et al.*, 2004] F. Bromberg, V. Honavar, and D. Caragea. Multi-agent data mining. In preparation, 2004.

[Buja and Lee, 2001] A. Buja and Y.S. Lee. Data mining criteria for tree-based regression and classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 27–36, San Francisco, CA, 2001. ACM Press.

[Cannataro and Talia, 2003] M. Cannataro and D. Talia. The Knowledge Grid. *Communications of the ACM*, 46(1):89–93, January 2003.

[Cannataro *et al.*, 2001] M. Cannataro, D. Talia, and P. Trunfio. Knowledge Grid: high performance knowledge discovery on the grid. In *Proceedings of the Second International Workshop on Grid Computing*, pages 38–50, Denver, CO, November 2001.

[Caragea *et al.*, 2000] D. Caragea, A. Silvescu, and V. Honavar. Agents that learn from distributed dynamic data sources. In *Proceedings of the Workshop on Learning Agents, Agents 2000/ECML 2000*, pages 53–61, Barcelona, Spain, 2000.

[Caragea *et al.*, 2001] D. Caragea, A. Silvescu, and V. Honavar. Invited chapter: Toward a theoretical framework for analysis and synthesis of agents that learn from distributed dynamic data sources. In S. Wermter, J. Austin, and D. Willshaw, editors, *Emerging Neural Architectures Based on Neuroscience*, volume 2036 of *Lecture Notes in Artificial Intelligence*, pages 547–559. Springer-Verlag, 2001.

[Caragea *et al.*, 2003] D. Caragea, A. Silvescu, and V. Honavar. Decision tree induction from distributed heterogeneous autonomous data sources. In *Proceedings of the International Conference on Intelligent Systems Design and Applications*, Tulsa, Oklahoma, 2003.

[Caragea *et al.*, 2004a] D. Caragea, J. Pathak, and V. Honavar. Query answering in indus. In preparation, 2004.

[Caragea *et al.*, 2004b] D. Caragea, J. Pathak, and V. Honavar. Statistical queries over semantically heterogeneous data sources. Submitted to the *Second International Workshop on Semantic Web and Databases, VLDB*, 2004.

[Caragea *et al.*, 2004c] D. Caragea, A. Silvescu, and V. Honavar. Characterization of sufficient statistics using fixed point equations. In preparation, 2004.

[Caragea *et al.*, 2004d] D. Caragea, A. Silvescu, and V. Honavar. A framework for learning from distributed data using sufficient statistics and its application to learning decision trees. *International Journal of Hybrid Intelligent Systems*, 1(2), 2004.

[Casella and Berger, 2001] G. Casella and R.L. Berger. *Statistical Inference*. Duxbury Press, Belmont, CA, 2001.

[Cesa-Bianchi *et al.*, 2001] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. In *Proceedings of the Neural Information Processing Systems Conference*, pages 359–366, Vancouver, British Columbia, Canada, December 2001. MIT Press.

[Chan *et al.*, 1999] P. Chan, W. Fan, A. Prodromidis, and S. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems*, pages 67–74, Nov/Dec 1999.

[Chang and Garcia-Molina, 1999] C. K. Chang and H. Garcia-Molina. Mind your vocabulary: query mapping across heterogeneous information sources. In *ACM SIGMOD International Conference On Management of Data*, Philadelphia, PA, June 1999.

[Chattratichat *et al.*, 1999] J. Chattratichat, J. Darlington, Y. Guo, S. Hedvall, M. Koler, and J. Syed. An architecture for distributed enterprise data mining. In *High Performance Computing Networking*, pages 573–582, Amsterdam, Netherlands, 1999.

[Chen and Krishnamoorthy, 2002] R. Chen and S. Krishnamoorthy. A new algorithm for learning parameters of a Bayesian network from distributed data. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 585–588, Maebashi City, Japan, December 2002. IEEE Computer Society.

[Chen *et al.*, 2001] R. Chen, S. Krishnamoorthy, and H. Kargupta. Distributed Web mining using Bayesian networks from multiple data streams. In *Proceedings of the IEEE International Conference on Data Mining*, pages 281–288. IEEE Press, November 2001.

[Chen *et al.*, 2003a] J. Chen, S. Chung, and L. Wong. The Kleisli query system as a backbone for bioinformatics data integration and analisis. *Bioinformatics*, pages 147–188, 2003.

[Chen *et al.*, 2003b] R. Chen, K. Sivakumar, and H. Kargupta. Distributed Bayesian mining from heterogeneous data. *Knowledge and Information Systems Journal*, 2003.

[Chervenak *et al.*, 1999] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 1999.

[Clifton *et al.*, 2002] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, , and M. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations*, 4(2), December 2002.

[Cook and Holder, 2000] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[Cortes and Vapnik, 1995] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[Cover and Hart, 1967] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[Cristianini and Shawe-Taylor, 2000] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[Curcin *et al.*, 2002] V. Curcin, M. Ghanem, Y. Guo, M. Köhler, A. Rowe, J. Syed, and P. Wendel. Discovery Net: towards a grid of knowledge discovery. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 658–663, Edmonton, Canada, 2002. ACM Press.

[Darlington, 1990] R. B. Darlington. *Regression and linear models*. McGraw-Hill, 1990.

[Davidson *et al.*, 2001] S. Davidson, J. Crabtree, B. Brunk, J. Schug, V. Tannen, G. Overton, and C. Stoeckert. K2/Kleisli and GUS: experiments in integrated access to genomic data sources. *IBM Journal*, 40(2), 2001.

[Dhillon and Modha, 1999] I. Dhillon and D. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Proceedings of the KDD'99 Workshop on High Performance Knowledge Discovery*, pages 245–260, San Diego, CA, August 1999.

[Dietterich, 2000] T.G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

[Domingos, 1997] P. Domingos. Knowledge acquisition from examples via multiple models. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 98–106, Nashville, TN, 1997. Morgan Kaufmann.

[Draper *et al.*, 2001] D. Draper, A. Y. Halevy, and D. S. Weld. The nimble XML data integration system. In *ICDE*, pages 155–160, 2001.

[Du and Agrawal, 2002] W. Du and G. Agrawal. Using general grid tools and compiler technology for distributed data mining: Preliminary report. In *Proceedings of the Fifth International Workshop on High Performance Data Mining: Resource and Location Aware Mining (HPDM:RLM'02). In conjunction with Second International SIAM Conference on Data Mining*, Arlington, VA, April 2002.

[Du and Atallah, 2001] W. Du and M. J. Atallah. Privacy-preserving cooperative scientific computations. In *Proceedings of the Fourteenth IEEE Computer Security Foundations Workshop*, pages 273–282, Nova Scotia, Canada, June 2001.

[Du and Zhan, 2002] W. Du and Z. Zhan. Building decision tree classifier on private data. In *Proceedings of the Workshop on Privacy, Security, and Data Mining at The 2002 IEEE International Conference on Data Mining (ICDM'02)*, Maebashi City, Japan, December 2002.

[Duda *et al.*, 2000] R. Duda, E. Hart, and D. Stork. *Pattern Recognition.* Wiley, 2000.

[Dzeroski and Lavrac, 2001] S. Dzeroski and N Lavrac, editors. *Relational Data Mining.* Springer-Verlag, 2001.

[e-Science, 2001] The e-science initiative provides infrastructure to allow scientists to access very large data collections, and to use large-scale computing resources and high-performance visualization programs. See www.research-councils.ac.uk/escience (retreived 18 july 2004), 2001.

[Eckman, 2003] B. Eckman. A practitioner's guide to data management and data integration in bioinformatics. *Bioinformatics*, pages 3–74, 2003.

[Esposito *et al.*, 1997] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intellingence*, 19(5):476–491, 1997.

[Etzold *et al.*, 2003] T. Etzold, H. Harris, and S. Beulah. SRS: An integration platform for databanks and analysis tools in bioinformatics. *Bioinformatics Managing Scientific Data*, pages 35–74, 2003.

[Euroweb, 2002] *The Web and the GRID: from e-science to e-business. Workshop at Euroweb Conference*, Oxford, UK, December 2002.

[Farquhar *et al.*, 1996] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: a tool for collaborative ontology construction. In *Proceedings of the Knowledge Acquisition Workshops*, Stanford, CA, March 1996.

[Fayyad and Irani, 1992] U.M. Fayyad and K.B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8(1):87–102, 1992.

[Fern and Brodley, 2003] X. Fern and C. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML2003)*, Washington, DC, August 2003.

[Foti *et al.*, 2000] D. Foti, D. Lipari, C. Pizzuti, and D. Talia. Scalable parallel clustering for data mining on multicomputers. In *Proceedings of the Third Workshop on High Performance Data Mining. In conjunction with International Parallel and Distributed Processing Symposium 2000 (IPDPS'00)*, Cancun, Mexico, May 2000.

[Frchet, 1906] M. Frchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Mathematico di Palermo*, 22:1–74, 1906.

[Freund and Schapire, 1998] Y. Freund and R.E. Schapire. Large margin classification using the perceptron algorithm. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 209–217, Madison, Wisconsin, July 1998. ACM Press.

[Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1300–1309, Orlando, FL, July 1999. Morgan Kaufmann Publishers Inc.

[Friess *et al.*, 1998] T. Friess, N. Cristianini, and C. Campbell. The Kernel-Adatron: a fast and simple learning procedure for support vector machines. In J. Shavlik, editor, *Proceeding of the Fifteenth International Conference on Machine Learning (ICML)*, pages 188–196, Madison, Wisconsin, July 1998. Morgan Kaufmann Publishers Inc.

[Gansner and North, 2000] E.R. Gansner and S.C. North. An open graph visualization system and its applications to software engineering. *Software-Practice and Experience*, 30(11):1203–1233, 2000.

[Garcia-Molina *et al.*, 1997] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems, Special Issue on Next Generation Information Technologies and Systems*, 8(2), 1997.

[Gehrke *et al.*, 1999] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.Y. Loh. BOAT – optimistic decision tree construction. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, June 1999. ACM Press.

[Gehrke *et al.*, 2000] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest - a framework for fast decision tree construction of large datasets. *Data Mining and Knowledge Discovery*, 4(2/3):127–162, 2000.

[Getoor *et al.*, 2001] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Dzeroski and Eds. N. Lavrac, editors, *Relational Data Mining*. Springer-Verlag, 2001.

[Giannadakis *et al.*, 2003] N. Giannadakis, A. Rowe, M. Ghanem, and Y. Guo. Infogrid: providing information integration for knowledge discovery. *Information Sciences. Special Issue: Knowledge Discovery from Distributed Information Sources*, 155(3–4):199–226, October 2003.

[Gonzalez *et al.*, 2002] J.A. Gonzalez, L.B. Holder, and D.J. Cook. Graph-based relational concept learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 219–226, Sydney Australia, July 2002. Morgan Kaufmann Publishers Inc.

[Gradshteyn and Ryzhik, 1979] I.S. Gradshteyn and I.M. Ryzhik. *Tables of Integrals, Series, and Products, 5th ed.* Academic Press, 1979.

[Graefe *et al.*, 1998] G. Graefe, U. Fayyad, and S. Chaudhuri. On the efficient gathering of sufficient statistics for classification from large sql databases. In *Proceedings of the Fourth International Conference on KDD*, pages 204–208, Menlo Park, CA, 1998. AAAI Press.

[Graepel and Herbrich, 2000] Thore Graepel and Ralf Herbrich. From margin to sparsity. In *Proceedings of the Neural Information Processing Systems Conference*, pages 210–216. MIT Press, 2000.

[Grossman and Gou, 2001] L.R. Grossman and Y. Gou. Parallel methods for scaling data mining algorithms to large data sets. In J.M. Zytkow, editor, *Handbook on Data Mining and Knowledge Discovery*. Oxford University Press, 2001.

[Grossman *et al.*, 2000] R. L. Grossman, S. Bailey, A. Ramu, B. Malhi, and A. Turinsky. The preliminary design of Papyrus: a system for high performance, distributed data mining over clusters. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 259–275. MIT Press, 2000.

[Gruber and Wills, 1993] P.M. Gruber and J.M. Wills. *Handbook of Convex Geometry*. Elsevier Science Publishers B.V., 1993.

[Guo, 2003] Y. Guo. Discovery Net: a UK e-science pilot project for grid based knowledge discovery service. In *Proceedings of the Workshop on Data Mining and Exploration Middleware for Distributed and Grid Computing*, Minneapolis, Minnesota, September 2003.

[Haas *et al.*, 1997] L.M. Haas, D. Kossmann, E. Wimmers, and J. Yan. Optimizing queries across diverse sources. In *Proceedings of the 23rd VLDB Conference*, pages 267–285, Athens, Greece, 1997.

[Haas *et al.*, 2001] L.M. Haas, P.M. Schwarz, P. Kodali, E. Kotlar, J.E. Rice, and W.P. Swope. DiscoveryLink: a system for integrated access to life sciences data sources. *IBM System Journal*, 40(2), 2001.

[Hall and Bowyer, 2003] L. Hall and K. Bowyer. Comparing pure parallel ensemble creation techniques against bagging. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, FL, November 2003.

[Haussler, 1988] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[Heflin and Hendler, 2000] J. Heflin and J. Hendler. Dynamic ontologies on the Web. In *Proceedings on the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 443–449, Austin, TX, July 2000.

[Heflin *et al.*, 1999] J. Heflin, J. Hendler, and S. Luke. Coping with changing ontologies in a distributed environment. In *Proceedings of the AAAI-99 Workshop on Ontology Management*, 1999.

[Hellerstein and Stonebraker, 1993] J.M. Hellerstein and M. Stonebraker. Predicate migration: optimizing queries with expensive predicates. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 267–276, Washington, DC, May 1993.

[Hendler, 2003] James Hendler. Science and the semantic web. *Science*, 299, January 2003.

[Honavar *et al.*, 1998] V. Honavar, L. Miller, and J.S. Wong. Distributed knowledge networks. In *Proceedings of the IEEE Conference on Information Technology*, Syracuse, NY, 1998. IEEE Press.

[Jaeger, 1997] M. Jaeger. Relational Bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-1997)*, Providence, Rhode Island, August 1997.

[Jennings and Wooldridge, 2001] N. Jennings and M. Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, *Handbook of Agent Technology*. MIT Press, 2001.

[Jensen, 2001] F. V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag, 2001.

[Jin and Agrawal, 2003] R. Jin and G. Agrawal. Communication and memory efficient parallel decision tree construction. In *Proceedings of the Third SIAM International Conference on Data Mining*, San Francisco, CA, May 2003.

[Joachims, 1999] T. Joachims. Making large-scale support vector machine learning practical. In Bernhard Schlkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 169–184. MIT Press, 1999.

[Jouve and Nicoloyannis, 2003] P.E. Jouve and N. Nicoloyannis. A new method for combining partitions, applications for cluster ensembles in KDD. In *Proceedings of Parallel and Distributed Computing for Machine Learning. In conjunction with the 14th European Conference on Machine Learning (ECML'03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, Cavtat-Dubrovnik, Croatia, September 2003.

[Kang *et al.*, 2004a] D.K. Kang, D. Fuller, and V. Honavar. Misuse and anomaly detection experiments on bag of system calls representation. In *Computer and Communications Security (CCS) Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC 2004)*, George W. Johnson Center at George Mason University, Fairfax, VA, USA, 2004. Held in conjunction with the Eleventh ACM Conference on Computer and Communications Security.

[Kang *et al.*, 2004b] D.K. Kang, A. Silvescu, and V. Honavar. Generation of attribute value taxonomies from data and their use in data-driven construction of accurate and compact classifiers, 2004. Submitted.

[Kantarcioglu and Clifton, 2002] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proceedings of ACM SIGMOD Workshop on Research Issues on DMKD'02*, June 2002.

[Kargupta *et al.*, 1997] H. Kargupta, I. Hamzaoglu, and B. Stafford. Scalable, distributed data mining using an agent based architecture. In D. Heckerman, H. Mannila, D. Pregibon,

and R. Uthurusamy, editors, *Proceedings of Knowledge Discovery And Data Mining*, pages 211–214, Menlo Park, CA, 1997. AAAI Press.

[Kargupta *et al.*, 1999] H. Kargupta, B.H. Park, D. Hershberger, and E. Johnson. Collective data mining: A new perspective toward distributed data mining. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*. MIT Press, 1999.

[Kargupta *et al.*, 2001] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson. Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, 3(4):422–448, 2001.

[Kargupta *et al.*, 2002] H. Kargupta, B.H. Park, S. Pittie, L. Liu, D. Kushraj, and K. Sarkar. Mobimine: monitoring the stock market from a pda. *SIGKDD Explorations Newsletter*, 3(2):37–46, 2002.

[Kargupta *et al.*, 2003] H. Kargupta, K. Liu, and J. Ryan. Random projection and privacy preserving correlation computation from distributed data. In *Proceedings of High Performance, Pervasive, and Data Stream Mining 6th International Workshop on High Performance Data Mining: Pervasive and Data Stream Mining (HPDM:PDS'03). In conjunction with Third International SIAM Conference on Data Mining*, San Francisco, CA, May 2003.

[Kearns, 1998] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998.

[Kersting and De Raedt, 2000] K. Kersting and L. De Raedt. Bayesian logic programs. In F. Furukawa, S. Muggleton, D. Michie, and L. De Raedt, editors, *Proceedings of the Seventeenth Machine Intelligence Workshop*, Bury St. Edmunds, Suffolk, U.K., 2000.

[Knobbe *et al.*, 1999] A.J. Knobbe, H. Blockeel, A. Siebes, and D.M.G. Van der Wallen. Multi-relational data mining. In *Benelearn '99*. Maastricht University, September 1999.

[Knoblock *et al.*, 2001] C.A. Knoblock, S. Minton, J.L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada. The ariadne approach to Web-based information integration. *International Journal of Cooperative Information Systems*, 10(1-2):145–169, 2001.

[Kohavi, 1996] R. Kohavi. Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, August 1996.

[Koller, 1999] D. Koller. Probabilistic relational models. In S. Dzeroski and P. Flach, editors, *Proceedings of Ninth International Workschop on Inductive Logic Programming (ILP-1999)*, number 1634 in LNAI, Bled, Slovenia, June 1999. Springer.

[Krishnaswamy *et al.*, 2002] S. Krishnaswamy, A. Zaslavsky, and W. S. Loke. Techniques for estimating the computation and communication costs of distributed data mining. In *Proceedings of International Conference on Computational Science (ICCS2002) - Part I*, volume 2331 of *Lecture Notes in Computer Science (LNCS)*, pages 603–612. Springer Verlag, 2002.

[Krishnaswamy *et al.*, 2003] S. Krishnaswamy, A. Zaslasvky, and W. S. Loke. Internet delivery of distributed data mining services: Architectures, issues and prospects. In V. K. Murthy and N. Shi, editors, *Architectural Issues of Web-enabled Electronic Business*, pages 113–127. Idea Group, 2003.

[Kuengkrai and Jaruskulchai, 2002] C. Kuengkrai and C. Jaruskulchai. A parallel learning algorithm for text classification. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, July 2002.

[Kumar, 2003] V. Kumar. Network intrusion detection using distributed data mining. In *Workshop on Data Mining and Exploration Middleware for Distributed and Grid Computing*, Minneapolis, MN, September 2003.

[Lambrecht *et al.*, 1999] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information-gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1204–1211. AAAI Press, 1999.

[Langford, May 2002] J. Langford. *Quantitatively Tight Sample Complexity Bounds*. PhD thesis, Computer Science, Carnegie Mellon University, May 2002.

[Leckie and Kotagiri, 2002] C. Leckie and R. Kotagiri. Learning to share distributed probabilistic beliefs. In *Proceedings of The Nineteenth International Conference on Machine Learning (ICML2002)*, Sydney, Australia, July 2002.

[Leiva *et al.*, 2002] H. Leiva, A. Atramentov, and V. Honavar. Experiments with MRDTL – a multi-relational decision tree learning algorithm. In Sašo Džeroski, Luc De Raedt, and Stefan Wrobel, editors, *MRDM02*, pages 97–112. University of Alberta, Edmonton, Canada, July 2002.

[Levy, 1998] A. Levy. The information manifold approach to data integration. *IEEE Intelligent Systems*, 13, 1998.

[Levy, 2000] A. Levy. Logic-based techniques in data integration. In *Logic-based artificial intelligence*, pages 575–595. Kluwer Academic Publishers, 2000.

[Lin *et al.*, 2002] C.-R. Lin, C.-H. Lee, M.-S.Chen, and P. S. Yu. Distributed data mining in a chain store database of short transactions. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 576–581, Edmonton, Canada, 2002. ACM Press.

[Lin *et al.*, 2004] X. Lin, C. Clifton, and M. Zhu. Privacy preserving clustering with distributed EM mixture modeling. *Knowledge and Information Systems*, 2004. To appear.

[Lindell and Pinkas, 2002] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002. An extended abstract appeared at the CRYPTO 2000 conference.

[Lindner and Morik, 1995] G. Lindner and K. Morik. Coupling a relational learning algorithm with a database system. In *Workshop Notes of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.

[Liu *et al.*, 2004] K. Liu, H. Kargupta, and J. Ryan. Distributed data mining bibliography, release 1.3, March 2004.

[Lu *et al.*, 1995] J. Lu, G. Moerkotte, J. Schue, and V.S. Subrahmanian. Efficient maintenance of materialized mediated views. In *Proceedings of 1995 ACM SIGMOD Conference on Management of Data*, San Jose, CA, 1995.

[Luenberger, 1973] D. Luenberger. *Introduction to linear and nonlinear programming.* Addison Wesley, 1973.

[Manning and Keane, 2001] A. M. Manning and J. A. Keane. Data allocation algorithm for parallel association rule discovery. In *Proceedings ot the Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2001)*, Hong Kong, China, April 2001.

[Mansour, 1994] J. Mansour. Learning boolean functions via the fourier transform. In *Theoretical Advances in Neural Computation and Learning.* Kluwer, 1994.

[McClean *et al.*, 2002] S. McClean, R. Páircéir, B. Scotney, and K. Greer. A negotiation agent for distributed heterogeneous statistical databases. *SSDBM 2002*, pages 207–216, 2002.

[McClean *et al.*, 2003] S. McClean, B. Scotney, and K. Greer. A scalable approach to integrating heterogeneous aggregate views of distributed databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, pages 232–235, 2003.

[McCulloch and Pitts, 1943] W. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[Merugu and Ghosh, 2003] S. Merugu and J. Ghosh. Privacy-preserving distributed clustering using generative models. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, FL, November 2003.

[Minksy and Papert, 1969] M. Minksy and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[Mitchell, 1997] T.M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MONET, 2004] MONET Consortium. *MONET Workshop: Mathematics on the Web*, University of Bath, 2004. See http://monet.nag.co.uk/cocoon/monet/index.html (retrieved 18 July 2004).

[Moore and Lee, 1998] Andrew W. Moore and Mary S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.

[Morinaga *et al.*, 2003] S. Morinaga, K. Yamanishi, and Jun ichi Takeuchi. Distributed cooperative mining for information consortium. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, August 2003.

[Muggleton, 1992] S. Muggleton. *Inductive Logic Programming*. Academic Press Ltd., 1992.

[Musen, 1998] M.A. Musen. Modern architectures for intelligent systems: Reusable ontologies and problem-solving methods. In C.G. Chute, editor, *AMIA Annual Symposium*, pages 46–52, Orlando FL, 1998.

[Ngo and Haddawy, 1997] L. Ngo and P. Haddawy. Answering queries form context sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 1997.

[Nilsson, 1965] N. J. Nilsson. *Learning Machines*. McGraw-Hill, 1965.

[Papageorgiou *et al.*, 2003] H. Papageorgiou, F. Pentaris, E. Theodorou, M. Vardaki, and M. Petrakos. A statistical metadata model for simultaneous manipulation of both data and metadata. *International Journal of Intelligent Systems*, 2003.

[Park and Kargupta, 2002a] B. Park and H. Kargupta. Constructing simpler decision trees from ensemble models using Fourier analysis. In *Proceedings of the 7th Workshop on Re-*

search Issues in Data Mining and Knowledge Discovery (DMKD'2002), pages 18–23, Madison, WI, June 2002. ACM SIGMOD.

[Park and Kargupta, 2002b] B. Park and H. Kargupta. Distributed data mining: algorithms, systems, and applications. In Nong Ye, editor, *Data Mining Handbook*, pages 341–358. IEA, 2002.

[Park *et al.*, 1995] J. S. Park, M.-S.Chen, and P. S. Yu. Efficient Parallel Data Mining for Association Rules. In *Proceedings of ACM International Conference on Information and Knowledge Management*, pages 31–36, Baltimore, MD, November 1995.

[Parthasarathy *et al.*, 2001] S. Parthasarathy, M. J. Zaki, M. Ogihara, and W. Li. Parallel data mining for association rules on shared-memory systems. *Knowledge and Information Systems*, 3(1):1–29, 2001.

[Pathak *et al.*, 2004] J. Pathak, D. Caragea, and V. Honavar. Ontology-extended component-based workfows - a framework for constructing complex workfows from semantically heterogeneous software components. In *IEEE International Conference on Information Integration and Reuse*, Las Vegas, Nevada, 2004. Submitted.

[Pearl, 2000] Judea Pearl. *Graphical Models for Probabilistic and Causal Reasoning.* Cambridge Press, 2000.

[Polikar *et al.*, 2001] R. Polikar, L. Udpa, S. Udpa, and V. Honavar. Learn++: an incremental learning algorithm for multi-layer perceptron networks. *IEEE Transactions on Systems, Man and Cybernetics*, 31(4):497–508, 2001.

[Poulet, 2003] F. Poulet. Multi-way Distributed SVM algorithms. In *Parallel and Distributed computing for Machine Learning. In conjunction with the 14th European Conference on Machine Learning (ECML'03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, Cavtat-Dubrovnik, Croatia, September 2003.

[Prodromidis *et al.*, 2000] A.L. Prodromidis, P. Chan, and S.J. Stolfo. Meta-learning in distributed data mining systems: issues and approaches. In H. Kargupta and P. Chan, editors, *Advances of Distributed Data Mining*. AAAI Press, 2000.

[Provost and Hennessy, 1996] F. Provost and D. Hennessy. Scaling up: distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[Provost and Kolluri, 1999] Foster J. Provost and Venkateswarlu Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3(2):131–169, 1999.

[Provost, 2000] F. Provost. Distributed data mining: Scaling up and beyond. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed Data Mining*. MIT Press, 2000.

[Quinlan, 1986] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Ramakrishanan and Gehrke, 2000] R. Ramakrishanan and J. Gehrke. *Database Management Systems. Second Edition.* The McGraw-Hill Companies, Inc., 2000.

[Rana *et al.*, 2000] O. Rana, D. Walker, M. Li, S. Lynden, and M. Ward. PaDDMAS: Parallel and Distributed Data Mining Application Suite. In *Fourteenth International Parallel and Distributed Processing Symposium*, pages 387–392, Cancun, Mexico, May 2000.

[RDF, 1995] Resource description framework, 1995. See http://www.w3.org/RDF (retrieved 18 July 2004).

[Reinoso-Castillo *et al.*, 2003] J. Reinoso-Castillo, A. Silvescu, D. Caragea, J. Pathak, and V. Honavar. Information extraction and integration from heterogeneous, distributed, autonomous information sources: a federated, query-centric approach. In *IEEE International Conference on Information Integration and Reuse*, Las Vegas, Nevada, November 2003.

[Reinoso-Castillo, 2002] J. Reinoso-Castillo. Ontology-driven query-centric federated solution for information extraction and integration from autonomous, heterogeneous, distributed

data sources. Masters dissertation, Department of Computer Science, Iowa State University, Ames, IA, 2002.

[Rissanen, 1978] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Rodriguez-Martinez and Roussopoulos, 2000] M. Rodriguez-Martinez and R. Roussopoulos. MOCHA: a self-extensible database middleware system for distributed data sources. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 213–224, Dallas, TX, 2000.

[Rosenblatt, 1958] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[Rumelhart *et al.*, 1986] D.E. Rumelhart, G.E. Hinton, and J.L. McClelland. A general framework for parallel distributed processing. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, 1986.

[Samatova *et al.*, 2002] N. F. Samatova, G. Ostrouchov, A. Geist, and A. Melechko. RACHET: an efficient cover-based merging of clustering hierarchies from distributed datasets. *Distributed and Parallel Databases*, 11(2):157–180, 2002.

[Sarawagi and Nagaralu, 2000] S. Sarawagi and S. H. Nagaralu. Data Mining Models as Services on the Internet. *SIGKDD Explorations*, 2(1):24–28, 2000.

[Scholkopf, 1997] B. Scholkopf. *Support Vector Learning*. Springer-Verlag, 1997.

[Schuster *et al.*, 2004] A. Schuster, R. Wolff, and B. Gilburd. Privacy-preserving association rule mining in large-scale distributed systems. In *Proceedings of Cluster Computing and the Grid (CCGrid)*, 2004.

[Shafer *et al.*, 1996] J.C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of 22th International Conference on VLDB, September 3-6, 1996, Mumbai (Bombay), India*. Morgan Kaufmann, 1996.

[Shek *et al.*, 1996] E. C. Shek, R. R. Muntz, E. Mesrobian, and K. W. Ng. Scalable exploratory data mining of distributed geoscientific datah. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 32–37, Portland, OR, 1996.

[Sheth and Larson, 1990] A. Sheth and J. Larson. Federated databases: architectures and issues. *ACM Computing Surveys*, 22(3):183–236, 1990.

[Silvescu *et al.*, 2004a] A. Silvescu, C. Andorf, D. Dobbs, and V. Honavar. Inter-element dependency models for sequence classification. In *IEEE International Conference on Data Mining*, 2004. Submitted.

[Silvescu *et al.*, 2004b] A. Silvescu, D. Caragea, O. Yakhnenko, D.K. Kang, J. Pathak, and V. Honavar. Data source independent learning algorithms through the means of sufficient statistics and data source wrappers. In Preparation, 2004.

[Sivakumar *et al.*, 2003] K. Sivakumar, R. Chen, and H. Kargupta. Learning Bayesian network structure from distributed data. In *Proceedings of the 3rd SIAM International Data Mining Conference*, pages 284–288, San Franciso, CA, May 2003.

[Skiena, 1997] S.S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 1997.

[Srivastava *et al.*, 1999] A. Srivastava, E. Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery*, 3(3):237–261, 1999.

[Stevens *et al.*, 2003] R. Stevens, C. Goble, N. Paton, S. Becchofer, G. Ng, P. Baker, and A. Bass. Complex query formulation over diverse sources in tambis. *Bioinformatics*, pages 189–220, 2003.

[Stolfo and others, 1997] S. Stolfo et al. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of Third International Conference on Knowledge Discovery and Data Mining*, pages 74–81, Menlo Park, CA, 1997. AAAI Press.

[Sunderam, 2003] V. Sunderam. Towards service-based approaches to data mining in grids. In *Workshop on Data Mining and Exploration Middleware for Distributed and Grid Computing*, Minneapolis, MN, September 2003.

[SWS, 2002] *Science on the Semantic Web Workshop: Building the Next Generation of Environmental Information Systems*, October 2002. See http://cimic.rutgers.edu/semantic/ (retrieved 18 July 2004).

[Syed *et al.*, 1999] N.A. Syed, H. Liu, and K.K. Sung. Incremental learning with support vector machines. In *Proceedings of the KDD Conference*, San Diego, CA, 1999.

[Szalay, 2001] A. S. Szalay, editor. *ASP Conference Series*, volume 238(3), 2001.

[Talia, 2003] D. Talia. Grid-based data mining and the knowledge grid framework. In *Workshop on Data Mining and Exploration Middleware for Distributed and Grid Computing*, Minneapolis, MN, September 2003.

[Tannen *et al.*, 2003] V. Tannen, S. Davidson, and S. Harker. The information integration in K2. *Bioinformatics*, pages 225–248, 2003.

[Tomasic *et al.*, 1998] A. Tomasic, L. Rashid, and P. Valduriez. Scaling heterogeneous databases and design of DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):808–823, 1998.

[Tsai *et al.*, 2001] H.J. Tsai, L.L. Miller, J. Xu, and S. Lin. Using ontologies to integrate domain specific data sources. In *ISCA 3rd International Conference on Information Reuse and Integration*, pages 62–67, Las Vegas, NV, 2001.

[Tsoumakas and Vlahavas, 2002] G. Tsoumakas and I. Vlahavas. Distributed data mining of large classifier ensembles. In *Proceedings of Companion Volume of the Second Hellenic Conference on Artificial Intelligence*, pages 249–256, Thessaloniki, Greece, April 2002.

[Tumer and Ghosh, 2000] K. Tumer and J. Ghosh. Robust order statistics based ensembles for distributed data mining. In Hillol Kargupta and Philip Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 185–210. MIT, 2000.

[Turinsky and Grossman, 2000] A. Turinsky and R. L. Grossman. A framework for finding distributed data mining strategies that are intermediate between centralized strategies and in-place strategies. In *Proceedings of KDD 2000 Workshop on Distributed Data Mining*, 2000.

[Tveit and Engum, 2003] A. Tveit and H. Engum. Parallelization of the Incremental Proximal Support Vector Machine Classifier using a Heap-based Tree Topology. In *Parallel and Distributed computing for Machine Learning. In conjunction with the 14th European Conference on Machine Learning (ECML'03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, Cavtat-Dubrovnik, Croatia, September 2003.

[Vaidya and Clifton, 2002] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, July 2002.

[Vaidya and Clifton, 2003] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, August 2003.

[Valiant, 1984] L.G. Valiant. A theory of the learnable. *Communication of the ACM*, 27(11):1134–1142, 1984.

[Vapnik and Chervonenkis, 1971] V.N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probaility and its Applications*, 16(2):264–280, 1971.

[Vapnik, 1998] V. Vapnik. *Statistical Learning Theory.* Springer-Verlag, 1998.

[Weiß, 1998] G. Weiß. A multiagent perspective of parallel and distributed machine learning. In K. P. Sycara and M. Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 226–230, New York, NY, 1998. ACM Press.

[Wiederhold and Genesereth, 1997] G. Wiederhold and M. Genesereth. The conceptual basis for mediation services. *IEEE Expert*, 12:38–47, 1997.

[Wirth *et al.*, 2001] R. Wirth, M. Borth, and J. Hipp. When distribution is part of the semantics: a new problem class for distributed knowledge discovery. In *Proceedings of PKDD-2001 Workshop on Ubiquitous Data Mining for Mobile and Distributed Environments*, pages 56–64, Freiburg, Germany, September 2001.

[Witten and Frank, 1999] I.H. Witten and E. Frank. *Data mining : practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 1999.

[Wolff *et al.*, 2003] R. Wolff, A. Schuster, and D. Trock. A high-performance distributed algorithm for mining association rules. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, FL, November 2003.

[Yan *et al.*, 2004a] C. Yan, D. Dobbs, and V. Honavar. A two-stage classifier for identification of protein-protein interface residues. *Bioinformatics*, 2004. In Press.

[Yan *et al.*, 2004b] C. Yan, V. Honavar, and D. Dobbs. Identifying protein-protein interaction sites from surface residues - a support vector machine approach. *Neural Computing Applications*, 2004. In press.

[Yang *et al.*, 1998] J. Yang, P. Pai, V. Honavar, and L. Miller. Mobile intelligent agents for document classification and retrieval: A machine learning approach. In *Proceedings of the European Symposium on Cybernetics and Systems Research*, 1998.

[Zaiane *et al.*, 2001] O. Zaiane, M. El-Hajj, and P. Lu. Fast parallel association rules mining without candidacy generation. In *IEEE 2001 International Conference on Data Mining (ICDM'2001)*, pages 665–668, 2001.

[Zaki, 1999] M. Zaki. Parallel and distributed association mining: a survey. *IEEE Concurrency*, 1999.

[Zhang and Honavar, 2003] J. Zhang and V. Honavar. Learning decision tree classifiers from attribute-value taxonomies and partially specified data. In T. Fawcett and N. Mishra,

editors, *Proceedings of the International Conference on Machine Learning*, pages 880–887, Washington, DC, 2003.

[Zhang and Honavar, 2004] J. Zhang and V. Honavar. Learning naive Bayes classifiers from attribute-value taxonomies and partially specified data. In *Proceedings of the Conference on Intelligent System Design and Applications*, In Press, 2004.

[Zhang *et al.*, 2003a] J. Zhang, R. Jin, Y. Yang, and A.G. Hauptmann. Modified logistic regression: An approximation to svm and its application in large-scale text categorization. In T. Fawcett and N. Mishra, editors, *Proceedings of the International Conference on Machine Learning*, Washington, DC, 2003.

[Zhang *et al.*, 2003b] J. Zhang, L. Miller, D. Cook, A. Hardjasamudra, and H. Hofman. Densityplot matrix display for large distributed data. In *Proceedings of the Third International Workshop on Visual Data Mining, Third IEEE International Conference on Data Mining*, Melbourne, FL, November 2003.