Learning Classifiers from Large Databases Using Statistical Queries

Neeraj Koul, Cornelia Caragea, and Vasant Honavar
Iowa State University, Ames -IA 50011
{neeraj,cornelia,honavar}@cs.iastate.edu

Vikas Bahirwani and Doina Caragea
Kansas State University, Manhattan, KS - 66506
{vikas,dcaragea }@ksu.edu

Abstract

[1] *We describe an approach to learning predictive models from large databases in settings where direct access to data is not available because of massive size of data, access restrictions, or bandwidth requirements. We outline some techniques for minimizing the number of statistical queries needed; and for efficiently coping with missing values in the data. We provide open source implementation of the decision tree and Naive bayes algorithms to demonstrate the feasibility of the proposed approach.*

1 Learning Using Statistical Queries

Advances in virtually every area of human endeavor are being increasingly driven by our ability to acquire knowledge from vast amounts of data. Most current approaches to learning from data assume direct access to data. However, in many practical applications, the large size, access restrictions, memory and bandwidth constraints, and in some instances, privacy considerations prohibit direct access to data. Hence, there is an urgent need for scalable approach to learning predictive models from large datasets (that cannot fit in the memory available on the device where the learning algorithm is executed). To address this need, especially in settings where the data reside in distributed repositories, Caragea et al. [3, 4] have introduced a general strategy for transforming a broad class of standard learning algorithms that assume in memory access to a dataset into algorithms that interact with the data source(s) only through statistical queries or procedures that can be executed on the remote data sources. This involves separating a learning algorithm into two components: (i) a statistical query [2] generation component that poses a set of statistical queries to be answered by a data source and (ii) a hypothesis construction component that uses the resulting statistics to modify a partially constructed hypothesis (and may further invoke the statistical query component as needed). The implementation of this strategy in practice requires effective methods for minimizing the cost of statistical queries, and for coping with missing values in data. This paper describes an approach to learning predictive models (e.g., decision trees) from large databases using statistical queries that is guaranteed to yield the same results as those obtained by the corresponding learning algorithms when they have direct access to data.

We assume that each data source D has an data descriptor $Desc(D)$ which describes the structure of the data (attributes and their domains) over which the predictive model is to be built. Formally $Desc(D) =< A, C, \mathcal{V} >$ where $A = \{a_1, a_2, \ldots, a_n\}$ is the set of attributes, $C \notin A$ a special attribute corresponding to the class label, $\mathcal{V} = \{V_{a_1}, \cdots V_{a_n}, V_C\}$ a set of *domains* where $V_{a_i} = \{v_1^i \cdots v_{m_i}^i\}$ is the set of possible values of attribute a_i ($m_i = |V_{a_i}|$) and V_C the set of possible class labels. To keep things simple, we assume that all the attributes are nominal. Given $Desc(D)$ the instance space $I = V_{a_1} \times V_{a_2} \times V_{a_3} \cdots V_{a_n}$. In this paper we assume that the data source is a relational database. $Desc(D)$ implicitly specifies the schema of the database as follows: the dataset is stored in a table named D and it has columns $\{a_1, a_2, \ldots, a_n\}$ corresponding to the attributes in A, and the column C corresponds to the class label. A dataset D is a multiset whose elements belong to $I \times V_C$. $Desc(\mathsf{D})$ is used to formulate the queries that are posed against the dataset D. Suppose the data source D supports a set of primitive queries Q_D. In our setting the primitive queries correspond to count queries against D. When D is a relational database, the count queries take the form: *Select Count(*) from D where C =* c_k *AND* $a_i = v_j^i$ represented as

[1] This research was supported in part by the grant IIS 0711356 from the National Science Foundation.

[2] A statistic is simply a function of a dataset; A statistical query returns a statistic (e.g., the number of instances in the dataset that have a specified value for a specified attribute.)

978-0-7695-3496-1/08 $25.00 © 2008 IEEE
DOI 10.1109/WIIAT.2008.366

$S(D, C = c_k, a_i = v_j^i)$.

We assume that the system expresses statistical queries against D in its own statistical query language Λ. A *query planner* Π that transforms a query $q(s_D)$ expressed in Λ for a statistic s_D into a plan for answering s_D using some subset of the primitive statistical queries Q_D. We assume that the query planner Π has at its disposal, a set of operators O that can be used to combine the answers to queries in Q_D to obtain a statistic s_D. In the case where Q_D correspond to count queries, O may include $+, -$. A *query plan* for s_D, denoted by $plan(s_D)$, is simply an *expression tree* that successively combines the answers to the primitive queries to obtain the answer to query s_D (expressed in the query language that is understood by the query planner): Each leaf nodes correspond to a *primitive query* in Q_D and each non-leaf nodes corresponds to an operator in O. We assume that the planner Π is guaranteed to produce a correct plan $plan(s_D)$ for every statistic s_D that is expressible in Λ. In general, there might be multiple query plans that can produce a given statistic s_D. For example, the plans $Select\ Count(*)\ from\ D\ where\ C = c_k$ and $\sum_{v_j^i \in V_i} S(D, C = c_k, a_i = v_j^i)$ yields the same statistic. While the first of these two plans may seem like the obvious one to choose, if all of answers to primitive queries used by the second plan are available to the system (perhaps because of other queries that have been executed and the results cached), it might be preferable to simply reuse the available results by choosing the second plan.

The learning algorithm L, when executed against a dataset D, generates at each step i, a *set* of statistical queries $S_i(D) = \{s_D(i,1) \cdots s_D(i,n_i)\}$ where each query in S_i is expressed in Λ. Let $Plan(S_i(D)) = \{plan(s_D(i,1)) \cdots plan(S_D(i,n_i))\}$ be the set of plans generated by the query planner for the set of queries $S_i(D)$. We denote by $Q(plan(s_D(i,j)))$, the set of the primitive queries used in the plan $plan(s_D(i,j))$. Note that $Q(Plan(S_i(D)))$ denotes the subset of *primitive* queries against D that to answer the set of queries $S_i(D)$. Let $Q(Plan(S_i(D))) = \sum_{j=1}^{n_i} Q(plan(s_D(i,j)))$. Let $Q^L = \sum_i Q(Plan(S_i(D)))$. Clearly, $\forall j\ Q(plan(s_D(i,j))) \subseteq Q(Plan(S_i(D))) \subseteq Q^L \subseteq Q_D$. Consider a sequence of sets of statistical queries $S_1(D) \cdots S_i(D)$ generated by L when it is executed against a dataset D. Let ϕ_i be the corresponding sequence of sets of query plans $Plan(S_1(D)), Plan(S_2(D)) \cdots Plan(S_i(D))$ produced by the query planner. Let $\hat{Q}(\phi_i) = \cup_{l=1}^{i} \hat{Q}(Plan(S_l(D)))$ denotes the set of primitive queries retrieved as a result. Assume that the system maintains a cache of answers to primitive queries that gets updated after each set of queries is answered during the execution of L against D. The contents of the cache after executing the query set $S_{i-1}(D)$ (but before executing $S_i(D)$) is simply $\hat{Q}(\phi_{i-1})$. Taking advantage of the cache, we have $\hat{Q}(Plan(S_i(D)) = Q(Plan(S_i(D))) - \hat{Q}(\phi_{i-1})$; That is, the set of primitive queries that need to be answered in executing the set of query plans $Plan(S_i(D))$ is precisely the set of primitive queries in $Q(Plan(S_i(D)))$ that are not already present in the cache prior to the generation of $Plan(S_i(D))$. Assuming that L generates a sequence of m query sets $S_1(D) \cdots S_m(D)$ prior to terminating with a learned hypothesis, we can define the *query complexity* of ϕ_m, denoted by $QC(\phi_m)$, as $\left|\hat{Q}(\phi_m)\right|$, that is the total number of primitive queries that are posed to the data source based on ϕ_m. The *communication complexity* of ϕ_m, denoted by $CC(\phi_i)$, is defined as the bandwidth needed to transmit the primitive queries (and retrieve answers from) to the data source D according to ϕ_m. The task of the query planner is to generate a sequence of sets of query plans ϕ_m so as to minimize the query complexity $QC(\phi_m)$ which can be important in settings where the data source imposes a cost for answering each primitive query or communication complexity $CC(\phi_m)$ (which can be important in settings where bandwidth is at a premium) or both. In addition to taking advantage of the cache as outlined above, at each step i, the set of plans $Plan(S_i(D)$ can be optimized by sharing primitive queries across the query plans for individual statistical queries in the query set $S_i(D)$.

2 Representative Learning Algorithms

Naive Bayes Learner

Naive Bayes [8] is a simple learning algorithm that often yields classifiers with satisfactory performance in many applications. Naive Bayes classifier assigns an instance $\mathbf{x} =< x_1 \cdots x_n >$ to the most probable class label under the assumption that the attributes of the instance are independent given the class:

$$C_{NB}(\mathbf{x}) = arg\ max_{c_k \in V_C}\ P(c_k) \prod_{j=1}^{n} P(x_i)|c_k)$$

During the learning phase, we need to estimate the class probabilities:

$$P(c_k) = \frac{S(D, C = c_k)}{S(D)}$$

and the probability of each possible value of each possible attribute for each class. That is probabilities of the form:

$$P(a_i = v_j^i | C = c_k) = \frac{S(D, C = c_k, a_i = v_j^i)}{S(D, C = c_k)}$$

where V_{a_i} denotes the domain of attribute a_i and $v_j^i \in V_{a_i}$ denotes the jth possible value in the domain V_{a_i} of the attribute a_i. Because learning Naive Bayes classifier requires, in the setting where the learner has direct access to the dataset, only a single pass through the dataset, in our setting, the learner needs to pose only a single set of queries against D, that is, a set of all queries of form $S(D), S(D, c_k)$ and $S(D, c_k, a_i = v^i_j)$. However, in the simple setting where the dataset D is known to contain no missing values, the query planner can exploit the fact that $S(D, c_k) = \sum_{v^i_j \in V_{a_i}} S(D, C = c_k, a_i = v^i_j)$ and that $S(D) = \sum_{c_k \in V_c} S(D, C = c_k)$ to generate query plans for the queries of the form $S(D)$ and $S(D, C = c_k)$ using answers to the queries of the form $S(D, C = c_k, a_i = v^i_j)$, so as to obtain a set of query plans with the minimal query complexity of $\sum_{i=1}^{|A|} |V_{a_i}||V_C|$. See section 3 for discussion of how to handle missing values in a learning Naive Bayes classifiers using statistical queries.

Decision Tree Learner

Decision Tree algorithms are among some of the most widely used machine learning algorithms for building classifiers from data. A decision tree learner recursively chooses at each step an attribute that yields the most information regarding the class label (e.g., as measured by the reduction in entropy). The choice of an attribute at a node in the decision tree partitions the dataset based on the values of the chosen attribute. This process is repeated until a desired termination criterion is satisfied. Hence, each path π from the root to a given node in the decision tree has associated with it, a subset of the data D^π. Extending the path π requires identifying an attribute that provides the maximal information about the class membership of instances in D^π. Given a dataset D^π the information gain for an attribute a_i, denoted by $Gain(D^\pi, a_i)$ is given by $H(D^\pi) - \sum_{a^i_j \in V_{a_i}} H(D^\pi_{a^i_j}) \times \frac{|D^\pi_{a^i_j}|}{|D^\pi|}$ where $D^\pi_{a^i_j}$ represents the sub data set of D^π where the attribute a_i takes the jth value (v^i_j) in its domain V_{a_i}. $H(D^\pi)$ denotes the *entropy* of the class distribution in $H(D^\pi)$[11]. We need to compute $Gain(D^\pi, a_i)$ using statistical queries against D. Let $S(D^\pi_{a_{ij}}, C = c_k, a_i = v^i_j)$ represent the count query over the dataset $D^\pi_{a^i_j}$ where the class label is c_k and the attribute a_i takes the value v^i_j. We have:

$$H(D^\pi_{a^i_j}) = -\sum_{c_k \in V_c} \frac{S(D^\pi, C = c_k, a_i = v^i_j)}{|D^\pi_{a^i_j}|} log_2 \frac{S(D^\pi, C = c_k, a_i = v^i_j)}{|D^\pi_{a^i_j}|}$$

$$H(D^\pi) = -\sum_{c_k \in V_c} \frac{S(D^\pi, C = c_k)}{|D^\pi|} log_2 \frac{S(D^\pi, C = c_k)}{|D^\pi|}$$

In the absence of missing values in the dataset D, we have:

$$|D^\pi_{a^i_j}| = \sum_{c_k \in V_c} S(D^\pi, C = c_k, a_i = v^i_j)$$

$$|D^\pi| = \sum_{v^i_j \in V_{a_i}} \sum_{c_k \in V_c} S(D^\pi, C = c_k, a_i = v^i_j)$$

$$S(D^\pi, C = c_k) = \sum_{v^i_j \in V_{a_i}} S(D^\pi, C = c_k, a_i = v^i_j)$$

Hence, in absence of missing values queries of the form $S(D^\pi, c_k, a_i = v^i_j)$ over $Desc(D^\pi)$ suffice to choose the attribute to be chosen for the next node that extends the path π. However, because $Desc(D^\pi)$ is not available to the system, we need to compute it from $Desc(D)$ and the path π. Initially, the path π is empty, and the corresponding $Desc(D^\phi) = Desc(D) =< A, C, \mathcal{V} >$. Consider a path π which is a one-step extension of a path ψ and obtained by appending an arc $a_i = v^i_j$ to ψ. Then $Desc^\pi =< A_\pi, C, \mathcal{V}_\pi >$ where $A_\pi = A_\psi - \{a_i\}$ and $\mathcal{V}_\pi = \mathcal{V}_\psi - \{V_i\}$. It is easy to see that a query q over D^π can be expressed in terms of a query over D by simply adding to q, a clause $Clause(\pi)$ that corresponds to the values associated with each of the attributes along the path π. For example, $S(D^\pi, C = c_k, a_j = v^j_l)$ as *Select Count(*) From D Where* $C = c_k$ *AND* $a_j = v^j_l$ *AND* $Clause(\pi)$. Note the resulting query is a query against the data set D.

The query complexity of the optimal sequence of sets of query plans ϕ for constructing a decision tree over a dataset D can be derived by noting that in presence of no missing values, extending a path π requires answers to all queries of the $S(D^\pi, c_k, a_i = v^i_j)$ over $Desc(D^\pi)$. To simplify the calculation of the query complexity, assume that each of the attributes has a domain of the same cardinality. That is, $\forall i$, $|V_i| = m_i = m$. Suppose t_i is the total number of paths of length i in the decision tree $T(D)$ constructed from the dataset D. Then the query complexity of the optimal sequence of sets of queries posed by decision tree learner against a dataset D in constructing $T(D)$ is given by

$$\sum_{i=0}^{d-1} t_i \times (|A| - i) \times m \times |V_c|$$

where d is the length of the longest path in the decision tree.

3 Dealing with Missing Values

The presence of missing values for some of the attributes in some of the instances in the dataset D requires modifications to the basic procedures described above for learning using statistical queries. The techniques for dealing with missing values that have been well studied in the literature [7, 11] assume direct access to the dataset D. We assume that the database uses a designated special value (e.g., ?) to indicate a missing value. As an example, consider how to handle missing values in learning a decision tree under the assumption that the missing values only occur in one or more of the attributes but not the class label. Recall that for choosing the optimal attribute to extend a path π, the decision tree learner has to gather statistics over the dataset D^{π}. The number of instances in D^{π} that have missing values for the attribute a_i is given by:

$$\delta a_i = S(D^{\pi}) - \sum_{ck \in V_c} \sum_{v_j^i \in V_{a_i}} S(c_k, a_i = v_j^i)$$

The calculation of δa_i does not entail any queries beyond those required under the assumption of no missing values which include queries of the form $S(D^{\pi})$ and $S(D^{\pi}, C = c_k, a_i = v_j^i)$. [3]

Distributing counts for missing values according to the observed distribution of the attribute values is one way to handle missing values. Once we have obtained the counts from the data set D^{π} we modify all the counts of form $S(D^{\pi}, C = c_k, a_i = v_j^i)$ by adding

$$\delta a_i \times \frac{S(D^{\pi}, C = c_k, a_i = v_j^i)}{\sum_{v_j^i \in V_{a_i}} S(D^{\pi}, C = c_k, a_i = v_j^i)}$$

This approach can be extended in a relatively straightforward manner to deal with missing class labels. In the case of Naive Bayes, the procedure for handling missing attribute values is similar, with with D_{π} replaced by D.

4 Results and Discussion

We have completed an open source implementation in Java as part of the INDUS Data Mining Toolkit. The system also includes some utilities that support reading the data from an arff file format that is used in WEKA [12]. Unlike in the case of WEKA implementation of popular learning algorithms which require the entire dataset to be read into memory, the sufficient statistics based approach to learning from data in INDUS interacts with the data source via statistical queries (and hence does not require access to data) and consequently supports learning classifiers from datasets that are too large to fit into the memory of the device on which the learning algorithm is executed.

Provost et al [10] survey work on scaling up learning algorithms. Examples of approaches that have been explored include parallelization of specific algorithms [6], support for disk resident data [1], and learning decision trees from statistical queries [9, 3, 2]. WekaDB [13] enables WEKA implementations of learning algorithms to be used with data that reside in a relational database. However, WekaDB does require access to the underlying dataset to read the instances and ability to modify the instances in the database to deal with missing values.

In contrast, the approach to learning classifiers from data using statistical queries described in this paper is designed for settings where the learning algorithm does not have direct access to the dataset in memory (because of the large size of the dataset, or bandwidth or privacy considerations) or ability to execute user defined code on the dataset. It can cope with learning scenarios in which some of the instances in the dataset have missing values for some of the attributes. We have also outlined some ideas for minimizing the query complexity, that is, the number of queries that have to be answered by the data source. Work in progress is aimed at augmenting the implementation to handle learning from distributed databases with disparate schema based on the general strategy outlined in [5].

References

[1] K. Alsabti, S. Ranka, and V. Singh. CLOUDS: A decision tree classifier for large datasets. In *Knowledge Discovery and Data Mining*, pages 2–8, 1998.

[2] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff. Hierarchical decision tree induction in distributed genomic databases. *IEEE Transactions on Knowledge and Data Engineering*, 17:1138–1151, 2005.

[3] D. Caragea. *Learning classifiers from distributed, semantically hetrogeneous, autonomous data sources*. PhD thesis, Iowa State University, 2004.

[4] D. Caragea, J. Zhang, J. Bao, J. Pathak, and V. Honavar. Algorithms and software for collaborative discovery from autonomous, semantically heterogeneous information sources (invited paper). In *Proceedings of the 16th International Conference on Algorithmic Learning Theory.*, volume 3734, pages 13–44. Springer-Verlag., 2005.

[5] V. Honavar and D. Caragea. *Next Generation of Data Mining*, chapter Towards Semantics-Enabled Infrastructure for Knowledge Acquisition from Distributed Data. Taylor and Francis, 2008.

[6] R. Jin and G. Agrawal. Communication and memory efficient parallel decision tree construction. 2003.

[7] W. Z. Liu, A. P. White, S. G. Thompson, and M. A. Bramer. Techniques for dealing with missing values in classification. *Lecture Notes in Computer Science*, 1280:527–??, 1997.

[8] T. M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[9] A. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67– 91, March 1998.

[10] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Min. Knowl. Discov.*, 3(2):131–169, 1999.

[11] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[12] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[13] B. Zou, X. Ma, B. Kemme, G. Newton, , and D. Precup. Data mining using relational database management systems. In *Advances in Knowledge Discovery and Data Mining*, volume 3918 of *Lecture Notes in Computer Science*, pages 657–667. Springer Berlin / Heidelberg, 2006.

[3] The $Clause(\pi)$ needs to be suitably modified to ensure that the sub dataset D^{π} does include those instances where the attributes that form the path π have a missing value. We omit the details due to space constraints.