Mining Massive Relational Databases

Geoff Hulten, Pedro Domingos, and Yeuhi Abe
Department of Computer Science and Engineering
University of Washington, Seattle, WA, 98195-2350
{ghulten, pedrod, yeuhi}@cs.washington.edu

Abstract

There is a large and growing mismatch between the size of the relational data sets available for mining and the amount of data our relational learning systems can process. In particular, most relational learning systems can operate on data sets containing thousands to tens of thousands of objects, while many real-world data sets grow at a rate of millions of objects a day. In this paper we explore the challenges that prevent relational learning systems from operating on massive data sets, and develop a learning system that overcomes some of them. Our system uses sampling, is efficient with disk accesses, and is able to learn from an order of magnitude more relational data than existing algorithms. We evaluate our system by using it to mine a collection of massive Web crawls, each containing millions of pages.

1 Introduction

Many researchers have found that the relations between the objects in a data set carry as much information about the domain as the properties of the objects themselves. This has lead to a great deal of interest in developing algorithms capable of explicitly learning from the relational structure in such data sets. Unfortunately, there is a wide and growing mismatch between the size of relational data sets available for mining and the size of relational data sets that our state of the art algorithms can process in a reasonable amount of time. In particular, most systems for learning complex models from relational data have been evaluated on data sets containing thousands to tens of thousands of objects, while many organizations today have data sets that grow at a rate of millions of objects a day. Thus we are not able to take full advantage of the available data.

There are several main challenges that must be met to allow our systems to run on modern data sets. Algorithmic complexity is one. A rule of thumb is that any learning algorithm with a complexity worse than $O(n \log n)$ (where n is the number of training samples) is unlikely to run on very large data sets in reasonable time. Unfortunately, the global nature of relational data (where each object is potentially related to every other object) often means the complexity of relational learning algorithms is considerably worse than this. Additionally, in some situations for example when learning from high speed, open ended data streams even $O(n)$ algorithms may not be sufficiently scalable. To address this, the most scalable propositional learning algorithms (for example BOAT [Gehrke *et al.*, 1999] and VFDT [Domingos and Hulten, 2000]) use sampling to decouple their runtimes from the size of training data. The scalability of these algorithms depends not on the amount of data available, but rather on the complexity of the concept being modeled. Unfortunately, it is difficult to sample relational data (see Jensen [1998] for a detailed discussion) and these propositional sampling techniques will need to be modified to work with relational data. Another scaling challenge is that many learning algorithms make essentially random access to training data. This is reasonable when data fits in RAM, but is prohibitive when it must be repeatedly swapped from disk, as is the case with large data sets. To address this, researchers have developed algorithms that carefully order their accesses to data on disk [Shafer *et al.*, 1996], that learn from summary structures instead of from data directly [Moore and Lee, 1997], or that work with a single scan over data. Unfortunately, it is not directly clear how these can be applied in relational settings. Another class of scaling challenges comes from the nature of the processes that generate large data sets. These processes exist over long periods of time and continuously generate data, and the distribution of this data often changes drastically as time goes by.

In previous work [Hulten and Domingos, 2002] we developed a framework capable of semi-automatically scaling up a wide class of propositional learning algorithms to address all of these challenges simultaneously. In the remainder of this paper we begin to extend our propositional scaling framework to the challenge of learning from massive relational data sets. In particular, we describe a system, called VFREL, which can learn from relational data sets containing millions of objects and relations. VFREL works by using sampling to help it very quickly identify the relations that are important to the learning task. It is then able to focus its attention on these important relations, while saving time (and data accesses) by ignoring ones that are not important. We evaluate our system by using it to build models for predicting the evolution of the Web, and mine a data set containing over a million Web pages, with millions of links among them.

In the next section we describe the form of the relational data our system works with. Following that we briefly review some of the methods currently used for relational learning and discuss the challenges to scaling them for very large data sets. The following section describes VFREL in detail. We then discuss our application and the experiments we conducted, and conclude.

2 Relational Data

We will now describe the form of the relational data that we mine. This formulation is similar to those given by Friedman *et al.* [1999] and by Jensen and Neville [2002c]. Data arrives as a set of *object sources*, each of which contains a set of *objects*. Object sources are typed, and thus each is restricted to contain objects conforming to a single *class*. It may be helpful to think of an object source as a table in a relational database, where each row in the table corresponds to an object. In the following discussion we will use X to refer to an object and $C(X)$ to refer to its class. Each class has a set of *intrinsic attributes*, and a set of *relations*. From these, a set of *relational attributes* is derived. We will describe each of these in turn.

Intrinsic attributes are properties of the objects in the domain. For example a *Product* object's attributes might include its price, description, weight, stock status, etc. Each attribute is either numeric or categorical. We denote the set of intrinsic attributes for $C(X)$ as $A(C(X))$ and X's intrinsic attribute named a as $X.a$.

Objects can be related to other objects. These relations are typed, and each relation has a source class and a destination class. *Following a relation* from an instance of the source class yields a (possibly empty) set of instances of the destination class. One critical feature of a relation is the cardinality of the set of objects that is reached by following it. If a relation always returns a single object it is called a *one-relation*; if the number of objects returned varies from object to object it is called a *many-relation*. Our notation for a relation r on class $C(X)$ is $C(X) \rightarrow r$. We denote the set of relations for $C(X)$ as $REL(C(X))$. We will use $X \Rightarrow r$ to denote the set of objects reached by following relation r from object X, and we will use $C(X) \Rightarrow r$ to denote the target class of the relation. The series of relations that are followed to get from one object to another is called a relational path. Also note that every relation has an inverse relation. For example, the inverse to the *Product* $\rightarrow$ *producedBy* relation is the *Manufacturer* $\rightarrow$ *produces* relation.

An object's *relational attributes* are logical attributes that contain information about the objects it is related to. For example, one of a *Product* object's relational attributes is the total number of products produced by its manufacturer. Relational attributes are defined recursively, and the relational attributes of an object consist of the intrinsic attributes and relational attributes of the objects it is related to, and so on. It is common to limit the depth of recursion in some manner.

Each object must have a fixed number of relational attributes for any given depth to facilitate the use of existing tools on relational data. Unfortunately each object with many-relations (or that is related to an object with many-relations) has a variable number of related objects for any given depth. In order to reconcile this difference, we aggregate the values of a set of instances into a fixed number of attributes using a set of aggregation functions. The attributes for any particular instance are a subset of the attributes that are possible at a class level (if a many-relation on an instance is empty, some of the class level attributes have no value for the instance). Thus, more formally, let $R(C, d)$ be the set of relational attributes for C up to a depth of d. Let the set of all attributes (intrinsic and relational) for the class to depth d be $ATT(C, d) = A(C) \cup R(C, d)$.

$$R(C,d) = \bigcup_{r \in REL(C)} \bigcup_{a \in ATT(C \Rightarrow r, d-1)} AGG(a) \quad (1)$$

When r is a one-relation AGG is the identity function. When r is a many-relation AGG applies a set of aggregation functions to a and results in one attribute per aggregation function. The aggregations used depend on the type of a; in our experiments we use min, max, mean, and standard deviation if a is numeric and mode if a is categorical. We also include one additional relational attribute per many-relation, which is the count of the number of objects that satisfy the relation. Each relational attribute uses an intrinsic attribute from a single class, and passes it through the set of aggregation functions for each many-relation between $C(X)$ and the class with the intrinsic attribute. For example, the relational attributes of *Manufacturer* might include the average price of products it produces, the maximum price of a product it produces, the count of the number of products it produces, the most common color of a product it produces, the maximum of the average sale price of products it produces, etc.

The definition of R above is at the class level, but we are interested in the values for these attributes at an instance level. This is simply a matter of starting from the instance, following relations and calculating aggregations as specified in the preceding definition. We describe this procedure in more detail (including pseudo-code) in Section 4.1.

3 Relational Learning

One of the possible goals for relational learning is to build models to predict the value of some *target attribute* (or attributes) of a *target class* (or classes) from the other attributes of the objects of the target class and the objects they are related to. (Note that the target attribute can be intrinsic or relational.) A training data set–with the values of the target attributes filled in–is presented to the learner, and the learner must produce a model that accurately predicts the values of the target attributes on some other data set, where they are unknown. This is the type of relational learning we will focus on in the remainder of this paper. Other possible goals for relational learning systems include building probabilistic models over link existence and object existence (see Getoor *et al.* [2001]).

Perhaps the simplest method for performing relational learning is to *flatten* the data into a propositional data set, and pass it to an existing propositional learning system. Flattening proceeds as follows: a target depth d is selected, and a

propositional training example is constructed for each object in the target source by calculating the values of the attributes in the set $ATT(C(X), d)$. The advantage of this method is its simplicity, but it has several disadvantages. One is that it is very slow: calculating the value for each attribute potentially requires loading a large portion of data set from disk, and, even for modest values of d, there can easily be thousands or tens of thousands of attributes for each object. This problem grows worse than linearly with the size of the relational data set, because larger data sets have more objects that need their attributes calculated, and each of these objects is related to more objects in the larger data set; the exact cost depends on the density of the relational structure in the data. Another disadvantage of this method is that it produces propositional learning problems with very large attribute spaces. Large attribute spaces lead many learning algorithms to overfit the training data. Further, this often means that the size of the flattened data set is much larger than the relational one, which leads to additional scaling challenges.

One method used to address these problems is to avoid flattening the entire database, and instead perform a search over the space of possible relational attributes. This is the method used by PRMs [Friedman *et al.*, 1999]. PRMs work by first selecting a small subset of the possible attributes using some form of feature selection. Sufficient statistics are gathered for the selected attributes and passed to a propositional learner (PRMs use a Bayesian Network learning algorithm, modified to learn coherent joint distributions in the presence relational data). When the learner produces a model, a new set of attributes is selected by performing another round of feature selection, taking into account the information contained in the partially learned model. The system gathers any new sufficient statistics it needs, and the propositional learner is called to refine its existing model with the new set of attributes. These steps are repeated until resources are exhausted or until the quality of the resulting model asymptotes.

These approaches improve on flattening, but they still do not scale to very large data sets. One reason is that they must flatten each attribute they are considering for every object in the target source before they can do any feature selection. This is wasteful because the feature selection task is often relatively easy, and decisions could be made much sooner with high confidence. Additionally, the greedy search procedures they use may miss interesting features that could be easily found with more systematic search. In the next section we will present our system, which addresses these problems.

4 Scaling Up Relational Learning

Our system, which we call VFREL, has three main components. The first is a query planner designed to provide efficient access to training data on disk as needed by the rest of our system. The second is a filter-based feature selection algorithm that is accelerated with sampling. The third is a propositional learning algorithm. At a high level, VFREL works by using sampling to select a promising subset of the possible relational attributes, saving time by flattening those while ignoring the others, and then calling a propositional learner on the flattened values. In particular, it begins by scanning a sample of the target objects and flattening all attributes up to a large depth. This is very slow, but VFREL only does it for a small sample of the target objects. It then pauses and uses statistical tests to identify attributes that are poor enough that, with high confidence, they would not be selected by the feature selection algorithm if it could see their values for all of the target objects. As soon as it identifies any such clear losers, VFREL saves time by eliminating them from further consideration. VFREL repeats this procedure, generating fewer and fewer attribute values (requiring fewer disk accesses and less processor time) on more and more of the data set. After computing attribute values for all of the target objects, VFREL performs a final round of feature selection, constructs a propositional data set from the final set of selected attributes, and passes it to a propositional learning algorithm. We will now describe the components of our system in more detail, starting with our data access module.

4.1 Efficient Data Access: Traversal Tree

VFREL needs to calculate the values of some relational attributes for each target object. In order to do this, every related object that is relevant to one of these attributes needs to be loaded from disk and processed. VFREL can determine which relations it needs to follow to gather this set of objects from the information it has at class level. It builds a tree of these required relational paths. It then traverses the tree, following each relation at most once, loading data into RAM only as it is needed, and computing the required attribute values. Traversal Trees work with binary relations. If the domain contains N-ary relations, they are encoded into binary relations in the usual way.

Nodes in the traversal tree correspond to classes, and edges correspond to relations. During its run, VFREL maintains a tree that contains exactly the relations that must be followed to calculate the values of the relational attributes of the target object that have not been determined to be clear losers. And so, at each node in the tree, VFREL maintains a list of the attributes whose values need to be calculated from the instances of that class that are encountered at that point in a traversal. In VFREL's first iteration the traversal tree is simply an unrolled version of the class graph, and can be constructed in time proportional to the size of $ATT(TargetClass, d)$ as follows. The root node represents the class of the target object. A child is added to this node for each class in $REL(TargetClass)$, and so on recursively until the tree is depth d. Let T be a node, T_c be the class represented by the node, e be an edge, and e_r be the relation represented by the edge. Next, we build a list on each node (let T_a be the list on node T) that represents the attribute values that must be calculated at that point in the traversal as follows. We compute the set $ATT(TargetClass, d)$. Each of these attributes is based on one of the attributes of one class (see Equation 1) and is added to the associated node's list. Following cycles in the relational structure can lead to some obviously redundant attributes. Many such attributes are removed at this point by removing length one cycles that involve a one-relation and its inverse.

When VFREL needs to calculate the value of the relational attributes for a particular target object it uses the traversal tree

Table 1: Pseudo-code for calculating attribute values with a traversal tree.

```
Procedure Traverse(T, O)
  T is a traversal tree
  O is an instance of class T_c
  Let R = {} be the results of the traversal
  Record in R the values for attributes in T_a from O
  For e ∈ Children(T)
    Let T^ch be the node reached via e
    Let Objs be X ⇒ e_r
    If Objs is empty, every attribute in T^ch and all of
      its children is missing, note this in R
    If e_r is a one relation, let O^ch be the object in Objs
      R = R ∪ Traverse(T^ch, O^ch)
    Else e_r is a many relation, let Tmp = {}
      For O^ch ∈ Objs
        Tmp = Tmp ∪ Traverse(T^ch, O^ch)
      Perform needed aggregations, note values in R
  Return R
```

to determine which objects to load from disk and when. Table 1 contains pseudo-code for the procedure.

As the run progresses, and attributes are eliminated by feature selection, VFREL will remove the eliminated attributes from the attribute lists on the traversal tree's nodes. Notice that a leaf with an empty attribute list corresponds to an object where every attribute has been determined to be a loser. Such objects do not need to be loaded from disk and so the leaf is pruned from the tree (internal nodes may have empty lists as they can still contribute through the objects that they are related to).

This traversal strategy allows VFREL to follow each edge in the traversal tree only once (instead of once per attribute, as might be done if following an edge required just a pointer dereference instead of a disk access).[1] It also allows VFREL to be very efficient with its RAM usage. In particular, at any point in the traversal it requires that one object be in RAM per edge in the path from the root to the current traversal tree node. It also requires RAM to store the partially computed attribute values. (The maximum space required for this is on the order of the number of relational attributes of the target class, since relational attribute values are computed in an online manner as objects are loaded from disk.) For each many-relation VFREL also maintains a list of hash keys for the objects it will need to load to finish following the relation.

4.2 Feature Selection with Sampling

Our system uses filter-based feature selection [Kononenko, 1994], [Kohavi and John, 1997] to explore the space of relational attributes. The goal is to identify the relational attributes that are most relevant to the learning task and accelerate our system by only calculating the values of these relevant attributes, while ignoring the rest. VFREL uses sampling to accelerate this process, and is able to eliminate attributes (and thus paths from the traversal tree) with less than one scan over the data set. This allows it to be more efficient than standard PRM learning.

Filter-based feature selection works as follows. The utility of each feature is estimated on training data with a scoring function (commonly information gain). The best N features are selected, and the rest are discarded. Traditionally, calculating the information gain of an attribute requires knowing the value of the attribute for every training example. In our context, this means that the entire data set needs to be flattened before feature selection can take place, which results in no speed gain. If we are willing to accept a small chance of making an error, we can use sampling to do much better. VFREL uses techniques developed by Hulten and Domingos [2002] and others to do just that. Standard statistical results can be used to obtain a high confidence bound on the difference between the gain observed for a feature on a sample of data and the true gain of the feature. For example, the Hoeffding bound [Hoeffding, 1963] says the following. Let x be a random variable with range R. Let $\hat{x}$ be the mean of n i.i.d. (independent and identically distributed) observations of x. Then, with probability $1-\delta$, the Hoeffding bound guarantees that $x > \hat{x} - \epsilon$ where

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{2}$$

We apply this bound to our setting as follows. Let $G(A_1, n)$ be the information gain observed for attribute A_1 on a sample of n examples and similarly for $G(A_2, n)$. Recall that the range of the information gain function is the log base two of the number of values of the target attribute. Let $\Delta = G(A_1, n) - G(A_2, 2)$. We bound Δ with the Hoeffding bound and thus, if $\Delta - \epsilon > 0$, we know with confidence $1-\delta$ that A_1 truly has a higher information gain than A_2, and thus that the feature selection algorithm would select A_1 over A_2 if the gains were computed from the entire training set, instead of from the sample. Thus, when trying to find the top N features in the training set, and after the values of relational attributes have been generated for the first n target objects, we can state the following. Let A_N be the attribute with the N^{th} best gain on the sample. Then, with confidence $1-\delta^*$, any attribute with a gain less than $G(A_N, n) - \epsilon$ is not one of the best N attributes. δ^* is different from the δ in the Hoeffding bound because many comparisons are involved in the feature selection, and thus the bound needs to be applied many times to assure a global level of confidence. We use a Bonferroni correction and set δ by dividing δ^*, the desired global confidence, by the number of bounds that need to hold during the algorithm's entire run.

Sampling from relational data may violate the i.i.d. requirement of the Hoeffding bound. Taking this into account, using non-i.i.d. extensions of Hoeffding-style bounds, is an important direction for future research (see also Jensen and Neville [2002a] [2002b]).

[1] Notice that the description here may require an object be loaded from disk more than once per traversal if it is reached via several different relational paths. The full VFREL system uses several forms of caching to reduce this redundancy, but they are not reported on or evaluated in this paper.

Table 2: Pseudo-code for the VFREL algorithm.

Let $F = ATT(TargetClass, d)$
Let T = Initial Traversal tree for F
Let D = Initial step size
Let C = Database cursor for the target object source
While C is not finished
  Calculate values for F on next D objects from C
  Compute information gain for attributes in F
  Order F by information gain
  Let G_N be gain of the N^{th} best attribute
  Remove from F every attribute with gain $< G_N - \epsilon$
  Update T by dropping the removed attributes
  Call the StepSize function to find next D

Return the result of the propositional learner on the
  best N attributes

4.3 The VFREL Algorithm

We will now describe VFREL, our algorithm for mining massive relational data sets, in detail. The inputs are a relational data set, a target class and target attribute of that class, a depth cutoff d, a global confidence target δ^*, a target number of features N, a function that specifies how many samples to take before performing a round of feature selection (StepSize below), and a propositional learning algorithm. Table 2 contains pseudo-code for VFREL.

VFREL iterates over the target objects and starts generating values for all of the attributes that are at most depth d away. It periodically pauses to perform a round of feature selection, informed by the data that has been generated up to that point. The information gain for each of the attributes being considered is computed, and they are sorted by their information gain. The N^{th} best attribute is determined, and its information gain is noted. From the Hoeffding bound, we know with high probability that any attribute with a score less than $G_N - \epsilon$ will not be selected as one of the final N attributes, and does not need to be considered further. In order to assure a global confidence of δ^* that the correct attributes are selected, each local ϵ is determined with $\delta = \delta^*/(|ATT(TargetClass, d)| * i)$, where i is an estimate of the total number of iterations of VFREL's main loop that will be performed[2]. When VFREL finishes with all the target objects, it performs one final round of feature selection, keeping only the top N features. Finally, a propositional data set is created from the attribute values that were calculated during the feature selection and the propositional learning algorithm is called to produce a model.

Notice that this algorithm assumes that objects are retrieved from the target object source in random order, which is usually possible. In our application, for example, we iterate over the keys of a DBM style hash table, which returns keys in essentially random order. Other settings may require a scan over the data set to randomize it.

Early iterations of VFREL take relatively long, as they generate values for many attributes, and thus require many objects be loaded from disk. As the algorithm proceeds, however, it is able to eliminate attributes that are clearly not going to be selected, stop following the relations associated with them, focus its attention on the promising attributes, and thus generate attribute values for later object much more quickly. VFREL will be most effective when there are many unpromising attributes that can be eliminated quickly, and when the promising attributes are all found along the same set of relational paths. In the next section we describe an application we developed to evaluate the performance of our algorithm, and to determine if it can successfully learn from massive relational data sets.

5 Predicting the Evolution of the Web

The World Wide Web has received much study in recent years. Researchers have studied ways to classify Web pages into categories (e.g., Slattery and Craven [2001]), search for high quality pages (e.g., Kleinberg [1998], Page *et al.* [1998]), model the way Web pages acquire links over time (e.g., Barabasi and Albert [2000], etc.) One commonality among much of this work is that analyzing the content of Web pages in isolation seldom produces the best results–the links between pages often contain critical information that must be taken into account. Unfortunately, as we have seen, state of the art systems for building complex relational models are in incapable of scaling to data sets the size of the Web.

In this section we describe an application of VFREL to mining a massive Web data set. The goal is to build a model that accurately predicts if a Web page's popularity will rise or fall in the future. Such models would be useful, for example, to help improve search engine results for new pages, and to help designers create pages that people will reference. We estimate the popularity change in a Web page by counting the number of pages that point to it in one crawl, and trying to predict if the page will be linked to by five or more additional pages, five or more fewer pages, or within five of the same number of pages in a future crawl. We take into account 47 intrinsic attributes of nearly two million Web pages. We also make use of relational information that includes seven object sources and millions of relations.

Our application begins with a crawl of approximately 1.7 million Web pages from .edu domains that was gathered in early June of 2001. The crawl contains pages from 31k unique Web hosts and uses 23 GB of disk space. It was gathered starting from a small set of seed Web pages (Google's top 20 results for the query 'university') and performing a breadth-first crawl until no more files would fit on the disk[3]. The crawl ran on a cluster of five 1 Ghz Linux machines, and took approximately 3 days to finish. We gathered a second crawl, using the same procedure and set of seed pages, in

[2]If the estimate is exceeded we report the global confidence that was actually achieved, or the algorithm can be run again with a better estimate if needed. Our experiments required just 13 iterations of the main loop.

[3]The version of Linux we used for these studies limited the number of files in a partition to 1.7 million. We plan on removing this limitation in a future study.

November of 2002. There were 563k pages that appeared in both crawls.

We put each of the pages that appeared in both crawls into a database (an object database which we implemented on top of GDBM). We used seven object sources to represent the domain, and their properties are as follows:

WebPage There are 563,083 Web page objects in our data set. Each has 47 attributes, including binary attributes to indicate the presence of the top 10 words according to information gain on the training set; the number of images; characterizations of alt text usage, script usage, color usage, etc.[4] and the PageRank [Page *et al.*, 1998] of the page within the subset of the Web covered by the first crawl.

WebPageLink There are 2,154,420 Web page link objects, one for each link between the pages in our data set. Each of these objects has a one-relation for its source and a one-relation for its destination.

Domain There are 21,069 domain objects in our data set. Each has a single categorical attribute, the Carnegie Classification (a publicly available classification of universities by their types) of the school it belongs to.

WebPageDomainLink There are 563,083 links from Web pages to their domain, one for each Web page. Each link has one numeric attribute, the depth of the page in the domain. Each also has a one-relation for the page and a one-relation for the domain.

Site We identified 412 sites in our crawl. A site is distinct from a domain by being managed by a small group of people and being about a well defined topic. We used a set of handcrafted regular expressions that examined URLs and identified sites including home pages, course pages, group pages, and project pages. The very low number of sites identified by our heuristics is problematic, and in future work we hope to improve this. Each site has an attribute that specifies its type.

WebPageSiteLink There were 1411 links between Web pages and sites. Each contains a one-relation to the page and a one-relation to the site.

SiteDomainLink There were 412 links from sites to their domains. Each has a single attribute, the depth of the site in the domain. Each also has a one-relation for the site and a one-relation for the domain.

Note that conceptually this domain could be modeled without the Link objects. We modeled it this way for several reasons: it is the best way to encode the many-many relation between WebPage objects in our database; it is conceptually simpler to have all links modeled the same way; it is cleaner and more extensible as we add additional features to the links (which we plan to do in future work); and it does not hurt efficiency compared to the other method.

We built index structures so that any relation could be followed by accessing the index on disk, and then loading the related objects from the GDBM on-disk hash table that contains them. The resulting database and associated index structures took on order of a day to construct on a 1Ghz PIII, and occupy approximately 900MB of disk space. Reading all the objects from disk in random order takes about 450 seconds. Notice that many of the attributes in our domain are numeric. We turn these attributes into categorical ones as needed by dividing the attribute into ten approximately equal-frequency regions. Each WebPage object has a target attribute, whose value is 'Gain5' if the number of links to the page in the new crawl is at least 5 greater than in the original crawl, 'Lose5' if the number of links to the page in the new crawl is at least 5 less than in the original crawl, and 'Same' otherwise. We evaluated the learning algorithms in this domain by removing the target attribute from a randomly selected 30% of the WebPage objects, using the learning algorithms to build models on the data set, and using the models to fill in these missing labels.

For these experiments we set VFREL's parameters as follows: maximum depth, $d = 5$; global confidence, $\delta^* = 5\%$; N, number of features to select $= 100$; and StepSize began at 1,000 and was doubled in every iteration where feature selection did not shrink the size of the traversal tree. We used the C4.5 decision tree learner [Quinlan, 1993] as the propositional learner. We selected this learner over a scalable propositional learner for two reasons: the N-attribute flattened training examples for the 563k Web page objects fit in RAM, and we wanted to make the contribution of our relational feature selection algorithm easier to evaluate. In future work we plan on combining VFREL with the scalable VFDT decision tree induction algorithm [Domingos and Hulten, 2000]. We ran our system in parallel on a cluster of five 1Ghz Pentium III workstations running Linux with RAM sizes ranging from 256MB to 512MB.

We compared our system to simply flattening the database and passing the flattened data to C4.5. With our computing resources we were able to flatten depths up to 2, and the flattened data sets are Flat0 (no relational attributes), Flat1, and Flat2 below. We also compared to one of the leading models of Web evolution, the preferential attachment model [Barabási *et al.*, 2000]. The preferential attachment model proposes that links are constantly being added to the Web, and that the probability that any particular page is the target of the next link is proportional to the number of links that it already has. We could not estimate the parameters needed to apply this model directly in our setting. Instead, we used the insight it is based on and built a decision tree on a single relational attribute: the number of pages that point to the target page (non-discretized).

We ran VFREL and Flat0-2 with all of their attributes (-full below) and also after performing additional feature selection to select the best 20 attributes in each setting. Table 3 contains the results of our experiments. Using 20 attributes yielded the best results for every system. VFREL with its 20 best attributes achieved the highest accuracy of any of the algorithms we considered. Note that while the differences in error rate are small on a percentage basis, they were measured on 169k test objects and represent real differences in performance. Also notice that increasing the depth of attributes

[4]Many of these attributes were gathered with the WebSAT toolkit: http://zing.ncsl.nist.gov/WebTools/

Table 3: Results of the comparison between VFREL, flattening depth 0 - 2, the preferential attachment model, and predicting the most common class, MCC, which was Same. We show VFREL and Flat0 - 2 with their full feature set and after doing additional feature selection.

Algorithm	Test Set Error	# Nodes	# Attribs
MCC	10.2%	0	0
PrefAtt	8.2%	5	1
Flat0	10.9%	18,372	20
Flat1	8.5%	11,663	20
Flat2	8.2%	9,741	20
VFREL	8.1%	7,465	20
Flat0-full	11.2%	10,197	47
Flat1-full	8.8%	15,117	50
Flat2-full	8.3%	10,308	330
VFREL-full	8.6%	14,289	100

considered results in smaller, more accurate models in our experiments. This suggests that these deeper attributes actually do contain valuable information for our task, and that it may be beneficial to explore further than depth 5 – we plan on doing this in future work. The runtimes for generating the flattened data sets were (in CPU + system hours): Flat0, .27; Flat1, .30; Flat2, 12.9. We estimate from generating the first 10k examples that Flat4 would have taken 54 days, and we estimate from generating the first 1,000 examples that Flat5 would have taken 261 days. Our system was able to generate values for the best 100 features up to depth 5 in 20 days of CPU time (4 days of wall time because it ran in parallel). VFREL is thus an order of magnitude faster than directly flattening the data, and produces the most accurate model of any of the systems we evaluated.

At the beginning of its run, VFREL was forced to follow 56 relational paths from each Web page to gather the objects needed to calculate the 3,536 attributes in $ATT(WebPage, 5)$ (after the obviously redundant ones were removed). By the end of the run it was following just 14 paths for each Web page. Every selected relational path begins by following the 'linked from' relation from the target object (that is, all selected relational attributes are properties of pages that point to the target page). After that, the 'links to', 'linked from', and 'domain' relations were used. None of the Site related attributes or relations were used to calculate the 100 best attribute values. We believe this will change when we improve our site identification heuristics.

The top 20 attributes included attributes formed using every aggregation we allowed except for mode. Eleven of them were aggregations of the PageRank of pages that pointed to the target, or were linked (in either direction) to pages that pointed to the target. Other selected features included aggregations of counts of Web pages, of depths of pages in their domain, of the number of words in link anchors, and of the size of related pages in bytes. The best attribute was the preferential attachment one, the count of the number of pages that point to the target. By examining the decision tree produced by C4.5 we determined that the information in the PageRank attributes was mostly captured by the preferential attachment attribute. We found the attributes that contributed to our system's improvement over the preferential attachment model were properties of other pages pointed to by the pages that pointed to the target, like the variance of the domain depths of the other pages pointed to by pages pointing to the target, and the popularity (as measured by the number of inlinks) of the other pages they point to. These features are a depth of 5 from the target class, and it is unlikely that they would have been found by other relational learning systems. We believe that properties of the pages pointing to the target are important for this prediction task because people find the target page (a prerequisite to linking to the page) through these links.

Generating attribute values for the median hundred Web pages out of the first thousand (before any feature selection) took 3,488 seconds and required that nearly 5 million object be loaded from disk. In the last iteration of VFREL's main loop, when it was exploring just 14 relational paths, the median 100 objects took just 153 seconds and 420k object loads – an improvement of an order of magnitude by either metric. There was a great deal of variance in the time it took to generate attributes for 100 objects. In fact, some single Web pages, even on the final iteration with only 14 relational paths, required thousands of seconds and millions of object loads. We examined some of these Web pages and found them to be extremely highly connected (tens of thousands of in links), on very large domains (with many tens of thousands of pages), or both. In future work we plan on exploring the use of online aggregations [Hellerstein *et al.*, 1997] to reduce the time needed to generate attribute values for these highly connected objects.

6 Related Work

Learning from relational data has been extensively studied by the inductive logic programming (ILP) community [Lavrac and Dzeroski, 1994]. In general, ILP learns models from a richer class than our work (for example, learning recursive concepts), but is also generally believed to be very inefficient for large databases. Blockeel *et al.* [1999] developed a scalable ILP system named TILDE that effectively flattens relational data into what they call interpretations and then uses a version of FOIL [Quinlan, 1990], modified to make efficient access to data from disk, on these interpretations. TILDE was evaluated on a synthetic data set with 100,000 training examples. VFREL scales to much larger data sets by using sampling to focus on relevant attributes and relations. Slattery and Craven [2001] extensively studied the use of relational learning for hyper-text documents. They developed a hybrid algorithm that combines Naive Bayes, FOIL, and many insights into the nature of the Web, and applied it to several Web mining tasks. Their focus, however, was not on scaling, and the largest data set they considered contained on the order of thousands of Web pages, while ours contains millions.

7 Future Work

Directions for future work on VFREL include: more closely integrating it with a scalable propositional learning algorithm

(for example VFDT); modifying learners to exploit information from the data generation process (for example, when a relation is missing, many related attributes simultaneously have missing values); extending it to the case where the contents of object sources change over time; modifying it to iterate between feature selection and learning phases; and applying it to other domains.

Future directions for our Web application include: performing a similar study with a larger Web crawl; performing a similar study on a more volatile portion of the Web (perhaps .com); adding more intrinsic attributes to the objects (words on links, more text, etc.); building models to predict which links will appear over time; and building models from the stream of pages that a crawler finds as it finds them.

8 Summary

In this paper we explored some of the issues that prevent relational learning algorithms from scaling to very large data sets. We developed a system, VFREL, which uses efficient data access and sampling to efficiently explore the space of relational attributes. We used VFREL to mine data sets containing millions of objects and links, and found it to build models that were more accurate than those produced by any of the systems we evaluated, discover novel relational attributes, and work an order of magnitude faster than the alternative approaches.

Acknowledgments

This research was partly supported by an NSF CAREER grant to the second author, by ONR grant no. N00014-02-1-0408, and by a gift from the Ford Motor Co.

References

[Barabási *et al.*, 2000] A. L. Barabási, R. Albert, and H. Jong. Scale-free characteristics of random networks: The topology of the World Wide Web. *Physica A*, 281:69–77, 2000.

[Blockeel *et al.*, 1999] H. Blockeel, L. D. Raedt, and N. Jacobs. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.

[Domingos and Hulten, 2000] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, 2000. 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 1300–1307, Stockholm, Sweden, 1999. Morgan Kaufmann.

[Gehrke *et al.*, 1999] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-L. Loh. BOAT: optimistic decision tree construction. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 169–180, Philadelphia, PA, 1999. ACM Press.

[Getoor *et al.*, 2001] L. Getoor, N. Friedman, D. Koller, and B. Tasker. Learning probabilistic models of relational structure. In *Proceedings of the 18th Intern. Conference on Machine Learning*, pages 170–177, San Francisco, CA, 2001. Morgan Kaufmann.

[Hellerstein *et al.*, 1997] J. Hellerstein, P. Hass, and H. Wang. Online aggregation. In *Proceedings of the SIGMOD Intern. Conferencce on Managment of Data*, Tuscon, AZ, 1997.

[Hoeffding, 1963] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[Hulten and Domingos, 2002] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 525–531, Edmonton, Canada, 2002. ACM Press.

[Jensen and Neville, 2002a] D. Jensen and J. Neville. Autocorrelation and linkage cause bias in evaluation of relational learners. In *Proceedings of the Twelfth International Conference on Inductive Logic Programming*, Sydney, Australia, 2002. Springer.

[Jensen and Neville, 2002b] D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 259–266, Sydney, Australia, 2002. Morgan Kaufmann.

[Jensen and Neville, 2002c] D. Jensen and J. Neville. Schemas and models. In *Proceedings of the Multi-Relational Data Mining Workshop, 8th SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining*, Edmonton, Canada, 2002. ACM Press.

[Jensen, 1998] D. Jensen. Statistical challenges to inductive inference in linked data. In *Preliminary papers of the 7th Intern. Workshop on Artificial Intelligence and Statistics*, 1998.

[Kleinberg, 1998] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, Baltimore, MD, 1998. ACM Press.

[Kohavi and John, 1997] R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2), 1997.

[Kononenko, 1994] I. Kononenko. Estimating attributes: Analysys and extensions of relief. In F. Bergadano and L. D. Raedt, editors, *Proceedings of the European Conference on Machine Learning*, 1994.

[Lavrac and Dzeroski, 1994] N. Lavrac and S. Dzeroski. *Inductive logic programming: techniques and applications*. Chichester, UK: Ellis Horwood, 1994.

[Moore and Lee, 1997] A. W. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1997.

[Page *et al.*, 1998] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University, Stanford, CA, 1998.

[Quinlan, 1990] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

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

[Shafer *et al.*, 1996] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the Twenty-Second International Conference on Very Large Databases*, pages 544–555, Bombay, India, 1996. Morgan Kaufmann.

[Slattery and Craven, 2001] S. Slattery and M. Craven. Relational learning with statistical predicate invention: better models for hypertext. *Machine Learning*, 43(1/2), 2001.