16 Feature Generation and Selection in Multi-Relational Statistical Learning

Alexandrin Popescul and Lyle H. Ungar

Using rich sets of features generated from relational data often improves the predictive accuracy of regression models. The number of feature candidates, however, rapidly grows prohibitively large as richer feature spaces are explored. We present a framework, structural generalized linear regression (SGLR), which flexibly integrates feature generation with model selection allowing (1) augmentation of relational representation with cluster-derived concepts, and (2) dynamic control over the search strategy used to generate features. Clustering increases the expressivity of feature spaces by creating new concepts which contribute to the creation of new features, and can lead to more accurate models. Dynamic feature generation, in which decisions of which features to generate are based on the results of run-time feature selection, can lead to the discovery of accurate models with significantly less computation than generating all features in advance. We present experimental results supporting these claims in two multirelational document mining applications: document classification and link prediction.

16.1 Introduction

We present a statistical relational learning method, structural generalized linear regression (SGLR), for building predictive regression models from relational databases or domains with implicit relational structure such as collections of documents linked by citations or hyperlinks. In SGRL, features are dynamically generated by a refinement-graph style search over SQL queries, and tested for potential inclusion into a generalized linear regression model, such as linear, logistic, or Poisson regression. This approach has several advantages over more traditional logic-based inductive logic programming (ILP) methods. The tables resulting from SQL queries are easily aggregated in many ways, giving a rich space of quantitative, as well as Boolean features. The resulting regression models are typically more accurate than

logical models. We also show how to automatically augment the original relational schema with additional derived features, facilitating the search for compound features.

SGLR, like several related methods [23, 18, 9, 27], searches a space of "feature generating expressions" to find those which generate new predictive features. In SGLR, a given relational database schema describing background data structures a search over database queries. Features are generated in two steps: a refinement-graph-like search of the space of SQL queries generates tables, which are then aggregated into real-valued features, which are tested for inclusion in a generalized linear model; i.e., each query generates a table, which in turn is aggregated to produce scalar feature candidates, from which statistically significant predictors are selected.

The initial relational schema is dynamically augmented with new relations containing concepts derived by clustering the data in the tables. For example, clustering documents by the words they contain or authors by venues they have published in gives new concepts – topics (document clusters) or communities (author clusters) – and new relations between the original items and the clusters they occur in (documents on a topic or authors in a community).

The main search is over the space of possible relational database queries, augmented to include aggregate or statistical operators, groupings, richer join conditions, and argmax-based queries. This search can be guided based on the types of predictive features discovered so far. We show below that a very simple "intelligent" search over the space of possible queries (and hence features) can result in discovery of predictive features with far less computation than static (e.g., breadth-first) search.

SGLR couples two elements helpful for successful learning:(1) a class of statistical models which outperforms logic-based models and (2) principled means of choosing what features to include in this model. Regression models are often more accurate than recursive partitioning methods such as C4.5 or FOIL-style logic descriptions. This difference is particularly apparent when there are vast numbers of potential features, many of which contribute some signal, for example, when words are included as features. Regression also allows us to use principled feature selection criteria such as Akaike information criterion (AIC), Bayes information criterion (BIC), and streaming feature selection (SFS) [4, 29, 33] to control against overfitting.

Figure 16.1 highlights the components of SGLR. Two main processes – relational feature generation and statistical modeling – are dynamically coupled into a single loop. Knowing the types of features selected so far by the statistical modeler allows the query generation component to guide its search, focusing on promising subspaces of the feature space. The search in the space of database queries involving one or more relations produces feature candidates one at a time for consideration by the statistical model selection component. The process results in a statistical model where each selected feature is the evaluation of a database query encoding a predictive data pattern in a given domain. We use logistic regression (or, equivalently, maximum entropy modeling). Features are tested sequentially for inclusion

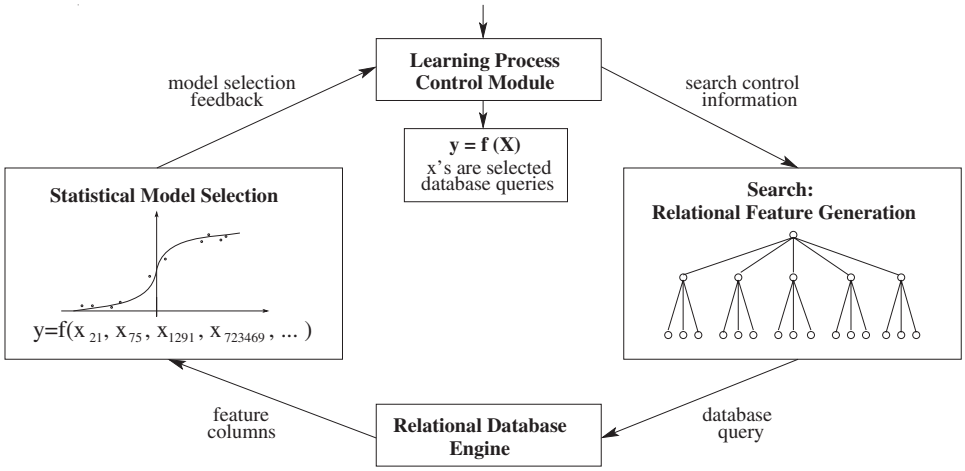

Figure 16.1 Learning process diagram.

in the regression model, and accepted if they are statistically significant after using a BIC [29] penalty to control against false discovery.

SGLR has several key characteristics which distinguish it from either pure probabilistic network modeling or ILP:

- The use of regression rather than logic allows the feature space to include statistical summaries or aggregates, and more expressive substitutions through nesting of intermediate aggregates (e.g., "How many times does this paper cite the most cited author in a conference to which it was submitted?").
- We use clustering to dynamically extend the set of relations generating new features. Clusters give better models of sparse data, improve scalability, and produce representations not possible with standard aggregates [12]. For example, one can cluster words based on co-occurrence in documents, giving "topics," or authors based on the number of papers they have published in the same venues, giving "communities." Once clusters are formed, they represent new relational concepts which are added to the relational database schema, and then used together with the original relations.
- We use relational database management systems and SQL rather than Prolog. Most real-world data lies in relational databases, with schemata and metainformation we can use. Relational database management systems incorporate decades of work on optimization, giving better scalability.
- Coupling generation and feature selection using discriminative modeling into a single loop gives a more flexible search than propositionalization Since the total number and type of features is not known in advance, the search formulation does lazy feature evaluation, allowing it to focus on more promising feature subspaces, giving higher time efficiency. Space efficiency is achieved by not storing pregenerated features, but rather considering them one by one as they are generated, and keeping only the few selected features.

We present results on two sets of tasks which use the data from CiteSeer (a.k.a. ResearchIndex), an online digital library of computer science papers [19]. CiteSeer contains a rich set of data, including paper titles, text of abstracts and documents, citation information, author names and affiliations, and conference or journal names. We represent CiteSeer as a relational database. For example, citation information is represented as a binary relation between citing and cited documents. Document authorship and publication venues of documents are also binary relations, while word counts can be represented as a ternary relation.

16.1.1 Invention of Cluster Relations

SGLR uses clustering to derive new relations and adds them to the database schema used in automatic generation of predictive features in statistical relational learning. Entities and relationships derived from clusters increase the expressivity of feature spaces by creating new first-class concepts. These concepts and relations are added to the database schema, and thus are considered (potentially in multiple combinations) during the search of the space of possible queries (figure 16.2). For example, in CiteSeer, papers can be clustered based on words or citations giving "topics," and authors can be clustered based on documents they coauthor giving "communities." In addition to simpler grouping (e.g., "Is this document on a given topic?"), such cluster-derived concepts become part of more complex feature expressions (e.g. "Does the database contain another document on the same topic and published in the same conference?"). The original database schema is implicitly used to decide which entities to cluster and what sources of attributes to use, possibly several per entity, creating alternative clusterings of the same objects. For example, documents can be clustered using words and, separately, using citations. Out of the large number of features generated, those which improve predictive accuracy are kept in the model, as decided by statistical feature selection criteria. Using cluster improves accuracy. Perhaps surprisingly, using cluster relations can also lead to a more rapid discovery of predictive features.

Cluster-relation invention as described here differs importantly from aggregation, which also creates new features from a relational representation [23, 26]. Aggregation allows one to summarize the information in a table returned from an SQL or logic query into scalar values usable by a regression model, for example, computing the *average* of a word count in all cited documents, or selecting a citing document with *max* number of incoming links. The clusters, on the other hand, create new relations in the database schema. The cluster relations are then used repeatedly to generate new queries and hence tables and features.

16.1.2 Dynamic Feature Generation

SGLR also supports *dynamic feature generation*, in which the order in which features are generated and evaluated is determined at run-time. Generating features is by far the most computationally demanding part of SGLR. In the example

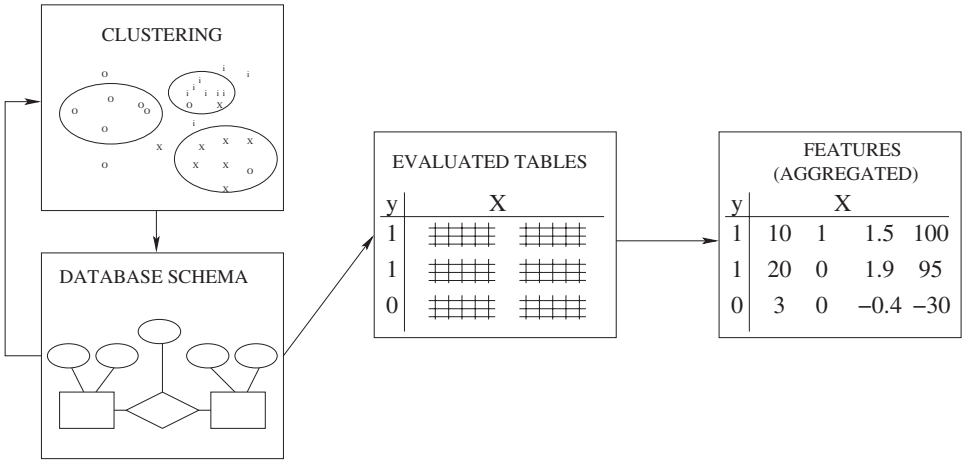

Figure 16.2 Cluster relations augment database schema used to produce feature candidates.

presented below, generating 100,000 features can take several CPU days due to the extensive SQL queries, particularly the joins. Dynamic feature generation can lead to discovery of predictive features with far less computation than generating all features in advance. When using the appropriate complexity penalties, one can still guarantee no overfitting, even when the order in which we generate features and test them for inclusion in the model is dynamically determined based on which features have so far been found to be predictive. This best first search often vastly reduces the number of computationally expensive feature evaluations.

Query expressions are assigned into multiple streams based on user-selected properties of the feature expressions; for example, based on the aggregate operator type. Since different sets of features are of different size (e.g., the number of different words is much greater than the number of journals or the number of topics), it is often easy to heuristically classify features into different streams. If feature generation has a known cost, this can also be taken into account. At each iteration, one of the streams is chosen to generate the next candidate feature, based on the expected utility of the streams features relative to those of other streams. For example, a simple and effective rule is to select the next query to be evaluated from the stream whose features have been included in the model in the highest percentage.

16.1.3 Chapter Overview

The following section describes the SGLR methodology in some detail, including how we cast feature generation as a search in the space of relational database queries, how cluster relations are created, and how the feature space is searched dynamically. Section 16.3 then describes two tasks using CiteSeer data which we use to test SGLR: classifying documents into their publication venues, conferences,

or journals, and predicting the existence of a citation between two documents. The tasks serve as a proxy for a more general problem of learning from inherently relational and noisy network data, including social networks. Relations between documents, cited documents, authors, publication venues, and text can all be explored to discover predictive features. We show that adding new relations to the database can improve accuracy, and that dynamic search can achieve the same accuracy as its static alternative while generating far fewer features, and hence reducing the required CPU time. The final two sections discuss SGLR in the context of related work, and summarize some of its advantages.

16.2 Detailed Methodology

As described above, SGLR dynamically couples two main components: generation of feature candidates via a search in the space of queries to a relational database, and their selection for inclusion in a regression model using statistical model selection criteria. First, we give the high-level SGLR algorithm. Lines in italics are the parts that do cluster-relation generation. We deliberately leave the stopping criterion underspecified. Given the incremental nature of model building in SGLR, deciding when to stop will often depend on the available CPU time and on the accuracy achieved so far.

1: **while** more features needed **do**
2: generate next SQL query expression using a refinement graph search
3: query the database and retrieve a table
4: *generate new cluster relations from the table*
5: *augment the database with derived relations*
6: apply aggregate operators to the table to produce a set of features
7: test the new features for inclusion in the model
8: **end while**

16.2.1 Notation and Basic Concepts

The language of nonrecursive first-order logic formulae maps directly into SQL and relational algebra, (see e.g., [8]). Our implementation uses SQL for efficiency and connectivity with relational database engines.

Throughout this paper we use the following schema:

$cites(FromDoc, ToDoc)$,
$author(Doc, Auth)$,
$published\_in(Doc, Venue)$,
$word\_count(Doc, Word, Int)$.

Domains, or types, used here are different from the primitive SQL types. The specification of these domains in addition to the primitive SQL types is necessary to guide the search process more efficiently.

First-order expressions are treated as database queries resulting in a table of all satisfying solutions, rather than a single Boolean value. The extended notation supports aggregation over entire query results or over their individual columns. Aggregate operators are subscripted with the corresponding variable name if applied to an individual column, or are used without subscripts if applied to the entire table. For example, an average count of the word "learning" in documents cited by document d, is denoted as:

$$class'(d) \sim ave_C\,[cites(d,\ D),\ word\_count(D,\ learning,\ C)],$$

where $\sim$ denotes "modeled using," i.e., *the right hand side of the expression is a feature* to be tested for inclusion in the regression model, the response variable, or target concept, on the left-hand side of the expression. We use *prime* with the target relation to avoid confusion with recursive queries when, as in the features below, a background relation instance and the target relation are of the same type.

An example of a feature useful for predicting a citation link between two documents $d1$ and $d2$ is the number of documents they both cite.

$$cites'(d1,\ d2) \sim count\,[cites(d1,\ D),\ cites(d2,\ D)]$$

The target concept is binary, and the feature (the right-hand side of the expression) is a database query about both target documents $d1$ and $d2$:

16.2.2 Feature Generation: Search in the Space of Database Queries

16.2.2.1 Refinement Graphs

Relational feature generation is a search problem. We use top-down search of refinement graphs [30, 11], supplemented with aggregate operators. Figure 16.3 shows a fragment of the search space in the domain of document classification. Each node is a database query about a learning example d, and evaluates to a table of all satisfying solutions. Aggregate operators are then applied to produce multiple features.

SGLR uses the following variation of the refinement operator: it forms a join of a given query with a relation from the database schema, expanding the query into the nodes covering *all* possible configurations of equality conditions involving attributes in the new relation, such that (1) each refinement contains at least one equality condition between a new and an old attribute, (2) any attribute can be set equal to a constant of its type, and (3) the types are used to avoid equalities between attributes of different types. This refinement operator is complete. Not all refinements produced are the most general refinements of a given query; however, we

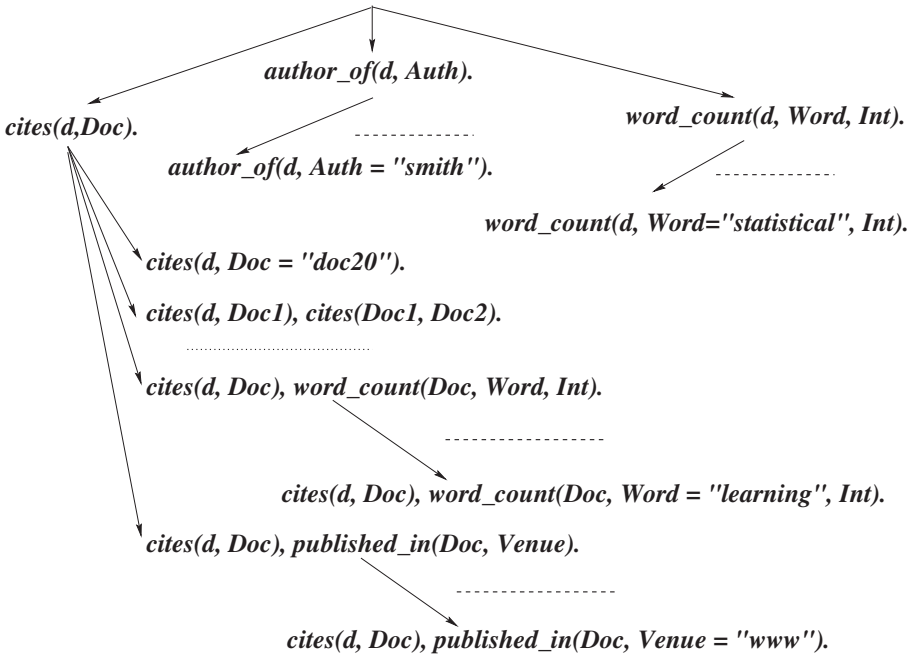

Figure 16.3 Fragment of the refinement graph in document classification domain

find that this definition can simplify pruning of equivalent subspaces by accounting only for the type and the number of relations in a query.

Table 16.1 presents the pseudocode of the refinement operator. After introducing notation with examples we walk over the pseudocode line by line. Evaluation of queries in the refinement graph nodes produces intermediate tables. Their aggregation is described in the next section.

R_i is a relation in a database schema $\mathbf{R}$; for example: $cites(doc1 : Document, doc2 : Document)$ is a binary relation in the database. Its two attributes are $doc1$ and $doc2$, both of type $Document$. Attributes in a relation are denoted by a subscripted letter A. A query to be refined is q, for example:

```
SELECT DISTINCT *
FROM cites R1, cites R2
WHERE R1.doc1='d1' AND R2.doc1='d2' AND R1.doc2=R2.doc2
```

Q_{ref} is a set of refinements of query q, the variable which accumulates the result of the refinement function. S_{eq} is a set of equality conditions which form the conjunction in the $WHERE$ clause of a refined query. $attrib(q)$ is a set of all attributes of relation instances in q:

```
{R1.doc1, R1.doc2, R2.doc1, R2.doc2}
```

$dom(A)$ are allowed constants of type A, e.g., containing all document IDs. $dom(A)$ must contain at least the target example constants identifying observations ($doc1$

Table 16.1 The refinement operator

refine(Query: q)

$Q_{ref} \leftarrow \{\}$
for each $R_i \in \mathbf{R}(i \in [1, n])$
  $S_{eq} \leftarrow \{\}$
  for each $A_j \in R_i$
    for each $A \in \{A_k | A_k \in attrib(q)\} \cup \{A_l | (A_l \text{ in } R_i) \wedge A_l \neq A_j\}$
      if($type(A_j) = type(A)$)
        $S_{eq} \leftarrow S_{eq} \cup \{norm(A_j = A)\}$
    for each $a \in dom(A_j)$
      $S_{eq} \leftarrow S_{eq} \cup \{norm(A_j = a)\}$
    for each $S \in 2^{S_{eq}}$
      if $\exists (A_i = A_j) \in S$ such that $A_i \in attrib(q) \vee A_j \in attrib(q)$
        $Q_{ref} \leftarrow Q_{ref} \cup \{q' | q'.WHERE =$
        $q.WHERE \cup \{S\} \wedge q'.FROM = q.FROM \cup \{R_i\}\}$
return Q_{ref}

and *doc*2, if they identify a target observation in the example above). In situations where generated features can include references to other constants, $dom(A)$ can include all values of A, or a subset, e.g., the entries with the highest correlation with the response variable or those above a count cutoff value. The following example of a query about the target pair $< d1, d2 >$ references other constants in the domain of document IDs; the query is nonempty when both $d1$ and $d2$ cite a particular document $d2370$:

```
SELECT DISTINCT *
FROM cites R1, cites R2
WHERE R1.doc1='d1' AND R2.doc1='d2' AND R1.doc2=R2.doc2
      AND R2.doc2='d2370'
```

$norm(A_i = A_j)$ alphanumerically orders A_i and A_j to avoid storing in S_{eq} equivalent entries $A_i = A_j$ and $A_j = A_i$. $type(A)$ is metatype of A, as is *Document* in the examples above, rather than an SQL type *String*. The set of equality conditions in query q is denoted by $q.WHERE$, e.g., a four-element set corresponding to the latter query example:

```
{R1.doc1='d1', R2.doc1='d2', R1.doc2=R2.doc2, R2.doc2='d2370'}
```

The refinement operator given in table 16.1 takes a query q as argument and returns the set of its refinements, Q_{ref}. Refinement of a given query starts by picking a relation instance in the database schema (loop starting at line 4). Adding this relation results in its Cartesian product with the view of q (not included in the

refinement set), which forms a "template" to be filled by allowed configurations of equality conditions. For each attribute A_j in the newly added relation R_i (starting at line 6) we find other attributes in q or in R_i itself, such that their equality with A_j can be included in the conjunction of equality conditions. This has to take into account metatypes (line 8), e.g., an attribute of type *Document* cannot be checked for equality with an attribute of type $Author : taskar$. Entries in S_{eq} are normalized (line 9) alphabetically to avoid equivalent entries. Equality of A_j with target example identifiers (constants) are added to the set of equality conditions S_{eq}, as well as possibly other constants of the type of A_j (lines 12-14). At this point S_{eq} contains all possible terms which can enter in the conjunction of refinements of q when being joined with R_i. A refined query is formed for each subset of S_{eq} (starting at line 15) such that at least one of the equality conditions in the subset (S) involves an attribute in q, i.e., an attribute of an "old" relation instance already present in q. The process repeats for all relations R_i in the schema $\mathbf{R}$ (back to line 4). A node resulting in an empty table for each observation is not refined any further since its refinements will be empty too.

16.2.2.2 Search Space Extension via Aggregate Operators

As in predicate calculus, aggregates are not part of the abstract relational languages. Practical systems, however, implement them as additional features. SQL supports the use aggregates which produce real values, rather than the more limited Boolean features produced by logic-based approaches. Regression modeling makes full use of these real-valued features.

As we described above, a node in our refinement graph is a query evaluating into a table. These tables are in turn aggregated by multiple operators to produce features. We use the aggregate operators common in relational language extensions: *count*, *ave*, *max*, and *min*; binary logic-style features are included through the *empty* aggregate operator. Aggregate operators are applied to an entire table or to its columns, as appropriate given type restrictions, e.g., *ave* cannot be applied to a column of a categorical type. When aggregate operators are not defined, e.g., the average of an empty set, we use an interaction with a 1/0 (defined/not-defined) indicator variable. Table 16.2 presents pseudocode of the aggregation procedure at each search node (called for each observation i).

The use of aggregate operators in feature generation complicates pruning of the search space. We use a hash function of partially evaluated feature columns to avoid fully recomputing equivalent features. In general, determining equivalence among relational expressions is known to be NP-complete, although polynomial algorithms exist for restricted classes of expressions; see, e.g., [3, 22]. Equivalence determination based on the homomorphism theorem for "tableau" query formalism, essentially the class of conjunctive queries we consider before aggregation, is given in [1]. Optimizations could be done by better avoiding generation of equivalent queries. Children nodes in the refinement graph can, of course, reuse evaluations

Table 16.2 Aggregation of refinement graph views

```
aggregate(View: v)

v is the evaluation of a search node query per observation i
    F ← {}
    for each Aggr_i ∈ A (i ∈ [1, n])      // A is a set of aggregate operators
        if(defined(Aggr_i(v))              // applicability of Aggr_i is determined by typing
            F ← F ∪ {Aggr_i(v)}    // e.g. "average" cannot be applied a categorical column
        for each column C ∈ v
            if(defined(Aggr_i(C)))
            F ← F ∪ {Aggr_i(C)}
return F
```

performed in their parent nodes; this considerably reduces computational burden at the expense of increased memory consumption.

16.3 Experimental Evaluation

16.3.1 Tasks and Data

We evaluate SGLR using data from CiteSeer (a.k.a. ResearchIndex), an online digital library of computer science papers [19]. CiteSeer includes text and titles of papers, citation information, author names, and conference or journal names. We represent CiteSeer as a relational database. For example, citation information is represented as a binary relation between citing and cited documents.

There are 1560 unique conferences and journals, 26,740 unique last names of authors, and 173,410 citations among our "universe" of 60,646 publication venues (conferences or journals) which could be extracted by matching with the DBLP database.[1] We limit the vocabulary to the 1000 most frequent words in the entire collection after Porter stemming and stop word removal to keep the data size to a manageable 6,894,712 `HasWord` relations denoting which words each document contains. These relations, and the number of instances are listed in table 16.3, along with the derived cluster relations which are later added to the database.

We explore two tasks using CiteSeer data: classifying documents into their publication venues and predicting the existence of a citation between two documents. The target concept pair in the two tasks are `<Document, Venue>` and `<Document, Document>` respectively. In both tasks, the search space contains queries based on

1. `http://dblp.uni-trier.de/`

Table 16.3 Sizes of the original and cluster-based relations

Relation	Size
PublishedIn(*doc*:Document, *vn*:Venue)	60,646
Author(*doc*:Document, *auth*:Person)	131,582
Citation(*from*:Document, *to*:Document)	173,410
HasWord(*doc*:Document, *word*:Word)	6,894,712
ClusterDocumentsByAuthors(*doc*:Document, *clust*:Clust0)	53,660
ClusterAuthorsByDocuments(*auth*:Person, *clust*:Clust1)	26,740
ClusterDocumentsByCitingDocuments(*doc*:Document,*clust*:Clust2)	31,603
ClusterDocumentsByCitedDocuments(*doc*:Document,*clust*:Clust3)	42,749
ClusterDocumentsByWords(*doc*:Document, *clust*:Clust4)	56,104
ClusterWordsByDocuments(*word*:Word, *clust*:Clust5)	1,000

several relations about documents, such as citation information, authorship, word content and publication venues of the document, and the response to be predicted is Boolean.

16.3.2 Cluster Creation

We use k-means (e.g., see [15]) to derive cluster relations; any other hard clustering algorithm could also be used. The results of clustering are represented by binary relations of the form `<ClusteredEntity, ClusterID>`.

Each many-to-many relation in the original schema can produce two distinct cluster relations (e.g., clusters of words by documents or of documents by words). Three out of the four relations in the schema presented above are many-to-many (`PublishedIn` is not); this results in six new cluster relations. Since the `PublishedIn` relation does not produce new clusters, nothing needs to be done to exclude the attributes of entities in the venue prediction training and test sets from participating in clustering. In link prediction, on the other hand, the relation corresponding to the target concept, `Citation`, does produce clusters, so in this case clustering is run without the links sampled for training and test sets.

k-means clustering requires the selection of k, the number of groups into which the entities are clustered. In the experiments presented here we fix k equal to 100 in all cluster relations except in `ClusterWordsByDocuments`, where only ten clusters were used because there are roughly an order of magnitude fewer clustered words than authors or documents. (This, since the vocabulary was limited to 1000 words.) The accuracy of resulting cluster-based models reported below could potentially be improved if one is willing to incur the cost of generating clusters with different values of k and testing the resulting features for model inclusion. One could also generate clusters from the rest of the tables generated as the space of queries is searched. For simplicity, we stuck to the first six such cluster relations. Table 16.3 summarizes the sizes of four original and the six derived cluster relations.

For clustering, we use the *tf-idf* vector-space cosine similarity [28]. The measure was originally designed for document similarity using word features, but we apply it here to broader types of data. In the formulae below, d stands for any object we want to cluster, and w are the attributes used to cluster d. For example, authors d can be clustered using the documents w they write. Below we refer to d's as documents and w's as words.

Each document d is viewed as a vector whose dimensions correspond to the words w's in the vocabulary; the vector elements are the *tf-idf* weights of the corresponding words, where $tfidf(w,d) = tf(w,d) \times idf(w)$. In the original formulation, "term frequency" $tf(w,d)$ is the number of times w occurs in d. In the experiments reported here we use *binary tf* indicating whether or not w occurs in d.[2] "Inverse document frequency" $idf(w) = \log \frac{|D|}{df(w)}$, where $|D|$ is the number of documents in a collection and $df(w)$ is the number of documents in which word w occurs at least once.

The similarity between two documents is then

$$sim(\mathbf{d_i}, \mathbf{d_j}) = \frac{\mathbf{d_i} \cdot \mathbf{d_j}}{||\mathbf{d_i}||||\mathbf{d_j}||},$$

where $\mathbf{d_i}$ and $\mathbf{d_j}$ are vectors with *tf-idf* coordinates as described above.

The cost of clustering is negligible compared to the cost of query evaluation, especially when one uses an efficient clustering algorithm. Linear-time algorithms are available using streaming methods [13], database methods [2, 32], or by exploiting regularities in the document citation structure [24].

16.3.3 Effect of Adding Cluster Relations

We compare models learned from the feature space generated from the four original noncluster relations with the models learned from the original four relations plus six derived cluster relations (`clustersNO` and `clustersYES` models). Models are learned with sequential feature selection using BIC [29], i.e., once each feature is generated, it is added to the model permanently if the BIC-penalized error improves, or is permanently dropped otherwise.

We use ten-fold reverse cross-validation to measure accuracy improvement from using cluster relations. All observations are split equally into ten sets. Each of the sets is used to train a model. Each of the models is tested on the remaining 90% of observations. This results in ten values per each tested level, which are used to derive error bounds. In venue prediction, there are 10,000 observations: 5000 positive examples of `<Document,Venue>` target pairs uniformly sampled from the relation `PublishedIn`, and 5000 negative examples where the document is uniformly sampled from the remaining documents, and the venue is uniformly

2. We use *binary tf* for consistency with the relation `HasWord`; we do not use counts in computing similarities since the original relation `HasWord` contains binary word occurrence data. Other derived cluster relations use naturally binary attributes.

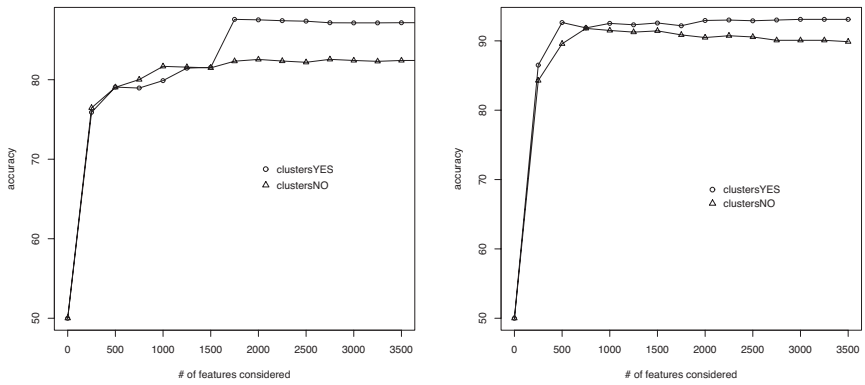

Figure 16.4 Learning curves: average test-set accuracy against the number of features generated from the training sets in ten-fold cross validation. *Left:* venue prediction, in each of ten runs $N_{train} = 1000$ and $N_{test} = 9000$. *Right:* link prediction, in each of ten runs $N_{train} = 500$ and $N_{test} = 4500$. Balanced positive/negative priors.

sampled from the domain of venues other than the true venue of the document. Positive example pairs are removed from the background relation `PublishedIn`, as well as the tuples involving documents sampled for the negative set. The size of the background relation `PublishedIn` decreases by 10,000 after removing training and test set tuples. In link prediction, the total number of observations is 5000: 2500 positive examples of `<Document,Document>` target pairs uniformly sampled from the `Citation` relation, and 2500 negative examples uniformly sampled from empty links in the citation graph. Positive example pairs are removed from the background relation `Citation`. The size of the background relation `Citation` reduces by 2500, the number of sampled positive examples.

A total of 3500 features are used in training each model. A numeric signature of partially evaluated features is maintained to avoid fully generating numerically equivalent features; note that this is different from avoiding syntactically equivalent nodes of the search space: two different queries can produce numerically equivalent feature columns, e.g., all zeros. Such repetition becomes common when feature generation progresses deeper in the search space.

Figure 16.4 presents test accuracy learning curves for models learned with and without cluster relations in venue prediction and link prediction respectively. Curve coordinates are averages over the runs in ten-fold cross validation. The learning curves show test-set accuracy changing with the number of features, in intervals of 250, generated and sequentially considered for model selection from the training set. The average test set accuracy of the cluster-based models after exploring the entire feature stream is 87.2% in venue prediction and 93.1% in link prediction, which is, respectively, 4.75 and 3.22 percentage points higher than the average accuracy of the models not using cluster relations.

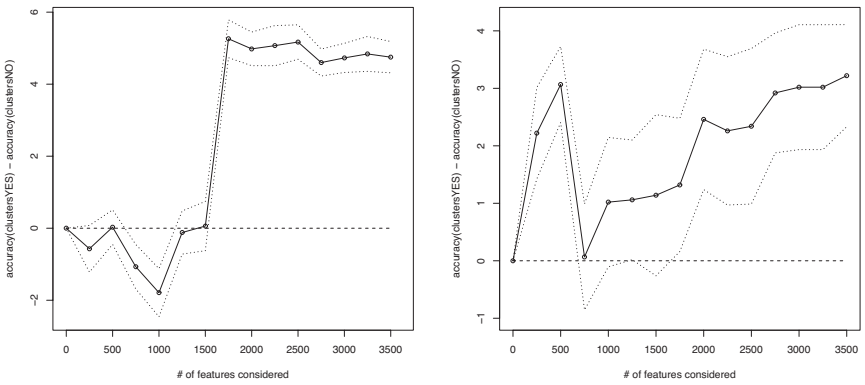

Figure 16.5 Mean accuracy difference: $accuracy(clustersYES) - accuracy(clustersNO)$ with 95% confidence intervals (bounds based on N=10 points, t-test distribution). *Left:* venue prediction. *Right:* link prediction

Figure 16.5 presents 95% confidence intervals of the difference in mean test accuracies of `clustersYES` and `clustersNO` models in venue prediction and link prediction respectively. In venue prediction, after exploring approximately half of the feature stream, the improvement in accuracy by the cluster-based models is statistically significant at the 95% confidence level according to the t-test (confidence intervals do not intersect with y=0). In the early feature generation, when considering the streams of about 1000 features, cluster-based models perform significantly worse: at this phase, additional cluster-based features, while not yet significantly improving accuracy, are delaying the discovery of significant noncluster-based features. In link prediction, while the significance of the improvement from cluster-based features is reduced early in the stream, it continuously increases throughout the rest of the stream. At the end of the stream the improvement in accuracy of the cluster-based model is 3.22 percentage points, statistically significant at the 99.8% confidence level. The highest accuracies (after seeing 750 features by `clustersNO` and after seeing 3500 features by `clustersYES`) also statistically differ: the accuracy improvement in cluster-based models is 1.49 percentage points, significant at the 99.9% confidence level.

The average number of features selected in ten `clustersYES` models is 32.0 in venue prediction and 32.3 in link prediction, respectively; 27.9 and 31.8 features on average were selected into `clustersNO` models from equally many feature candidates (3500). The BIC penalty used here allows a small amount of overfitting (see figure 16.4); more recent penalty methods such as SFS [33] avoid this problem.

The improved accuracy of the cluster-based model in venue prediction comes mostly from a single cluster-based feature. This feature was selected in all cross-validation runs. It is a binary cluster-based feature which is *on* for target document/venue pair `<D,V>`, if a document `D1` exists in the cluster where `D` belongs such that `D1` is published in the same venue as `D`. Using a logic-based notation, the

Table 16.4 Selected features which improve test accuracy by at least 1.0 percentage point. Target pair: `<D,V>`

FEATURE	MODEL
$size[publishedIn(\_, V)]$	BOTH
$exists[cites(D, D1), publishedIn(D1, V)]$	BOTH
$exists[cites(D1, D), publishedIn(D1, V)]$	BOTH
$exists[cites(D, D2), cites(D1, D2), publishedIn(D1, V)]$	BOTH
$exists[author(D, A), author(D1, A), publishedIn(D1, V)]$	BOTH
$exists[publishedIn(D1, V), docsByWords(D, C), docsByWords(D1, C)]$	CLUSTERSYES
$exists[cites(D, D3), cites(D3, D2), cites(D1, D2), publishedIn(D1, V)]$	CLUSTERSNO

feature is the following (abbreviated here `clustDocsByWords` by `topic`):[3]

$exists[publishedIn(D1, V), topic(D, C), topic(D1, C)]$.

The following three cluster-based features were selected in more than five cross-validation runs (nine, nine and six times respectively) in the link prediction task (target: `<D1,D2>`):
$exists[docsByCitedDocs(D1, C), docsByCitedDocs(D2, C)]$,
$exists[docsByWords(D1, C), docsByWords(D2, C)]$,
$exists[docsByCitingDocs(D1, C), docsByCitingDocs(D2, C)]$.

Table 16.4 gives examples of features which improved test accuracy by at least 1 percentage point over the previous state of the venue prediction model in one of the cross-validation runs. The first five features, in the generated order, are in both `clustersYES` and `clustersNO` models. `D` and `V` are respectively document and venue in the target pair `<D,V>`.

The features in table 16.4 can be summarized as follows: document D is more likely to appear in a conference or journal V, if venue V publishes a lot of papers; if document D cites or is cited by another document published in the same venue V; if, more generally, document D is close in the citation graph to other documents published in V; if the author of D published another paper in the same venue V; and finally, in the case of the cluster-based model, if another document on the latent topic of D and published in V exists.

The cluster relations shown above led to higher classification accuracy. Another potential advantage, not shown experimentally, is that cluster-based features are generally cheaper to generate, since cluster relations contain fewer tuples than the original relations from which they were derived. This can lead to reduced computational costs per number of generated feature candidates.

3. Note that `D1` is always distinct from `D` as the tuple with publication venue of document `D` is removed from the background relation `PublishedIn`.

16.3.4 Dynamic Feature Generation

Up to this point, we presented models learned when doing the breadth-first search of the feature space. In this section we explore an alternative search strategy in which separate streams are used to generate queries (and hence features), and new queries are preferentially selected from those streams which have been most productive of useful features. The database query evaluation used in feature generation dominates the computational cost of our statistical relational learning methodology; thus, intelligently deciding which queries to evaluate can have a significant effect on total cost.

Feature generation in the SGLR framework consists of two steps: query expression generation and query evaluation. The former is cheap as it involves only syntactic operations on query strings; the latter is computationally demanding. The experiment is set up to test two strategies which differ in the order in which queries are evaluated. In both strategies, query *expressions* are generated by breadth-first search. The base-line, static, strategy evaluates queries in the same order the expressions appear in the search queue, while the alternative, dynamic strategy, enqueues queries into separate streams at the time its expression is generated, but chooses the next feature to be evaluated from the stream with the highest ratio:

$$(featuresAdded + 1)/(featuresTried + 1),$$

where $featuresAdded$ is the number of features selected for addition to the model, and $featuresTried$ is the total number of features tried by feature selection in this stream. Many other ranking methods could be used; this one has the virtue of being simple and, for the realistic situation in which the density of predictive features tends to decrease as one goes far into a stream, complete.

In the tests below, we use two streams. The first stream contains queries with aggregate operators `exists` and `count` over the entire table. The second stream contains features which are the counts of unique elements in individual columns. We stop the experiment when one of the streams is exhausted.

We report the difference in test-set accuracy between dynamic and static feature generation. In each of four data sets, the difference in accuracy is plotted against the number of features evaluated and considered by feature selection We also kept track of the CPU time required for each of these cases. The MySQL database engine was used. Data sets 1, 2, and 3 took roughly 20,000 seconds, while data set 4 took 40,000 seconds. In all four cases, plots of accuracy vs. CPU time used were qualitatively similar to the plots shown in figure 16.6.

In the experiments presented here, one of the two feature streams was a clear winner, suggesting the heuristic splitting feature was effective. When the choice of a good heuristic is difficult, dynamic feature generation, in the worst case, will split features into "equally good" streams, and will asymptotically lead to the same expected performance as the static feature generation by taking features from different streams with equal likelihood.

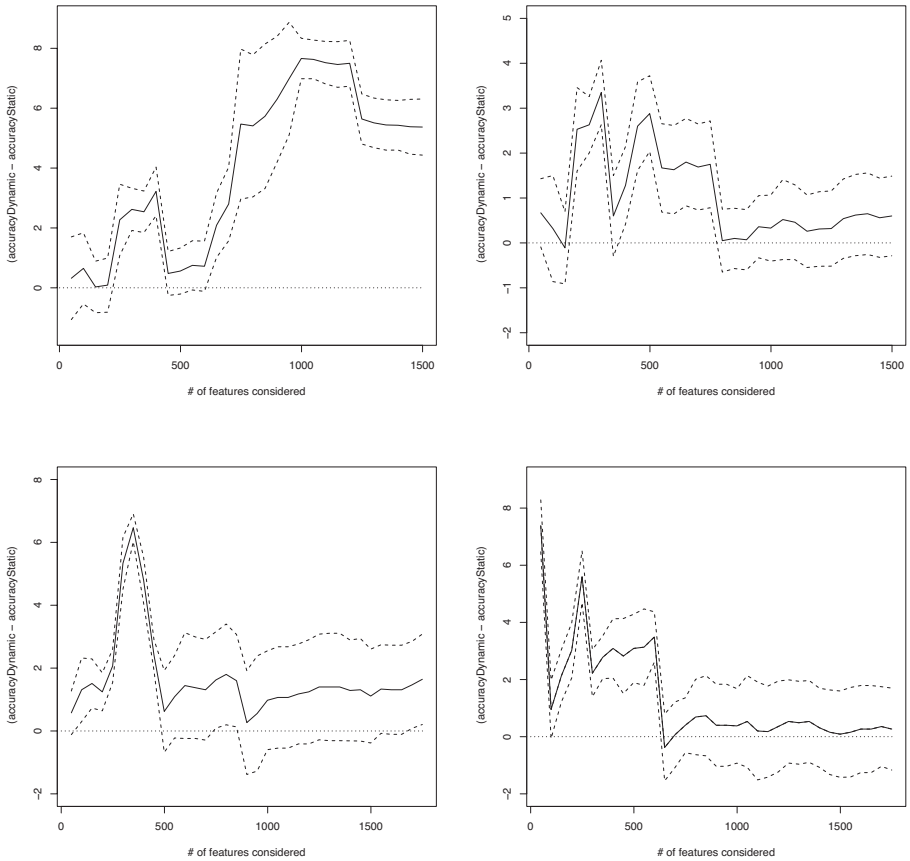

Figure 16.6 Test accuracy of the dynamic search minus test accuracy of the static search against the number of features considered. Errors: 95% confidence interval (ten-fold cross validation, in each of ten runs). In *row first* order: (a) Set 1 (venue prediction, with cluster relations), (b) set 2 (venue prediction, without cluster relations); (c) set 3 (link prediction, with cluster relations); (d) set 4 (link prediction, without cluster relations). $N_{train} = 1000$, $N_{test} = 9000$ in (a) and (b); $N_{train} = 500$, $N_{test} = 4500$ in (c) and (d)

The two-stream approach can be generalized to a multistream approach. For example, some streams can be formed based on the types of aggregate operators in query expressions, as we did here, and other streams can be formed based on the type of relations joined in a query, for example, split based on whether a query contains a cluster-derived relation, or a relation of the same type as the target concept. A query can be enqueued into one stream based on its aggregate operator, and into a different stream based on type of relation instances. The split of features into multiple streams need not be a disjoint partition. This method, if used with a check to avoid evaluation of a query which has been evaluated previously in a different stream, will not incur a significant increase in computational cost.

Another approach is to split features into multiple streams according to the sizes of their relation instances, which would serve as an estimate of evaluation time. This can lead to improvements for the following reasons: (1) out of two nearly collinear features a cheaper one will likely correspond to a simpler query and will be evaluated first, leading to approximately the same accuracy improvement as the second more expensive feature; and (2) there is no obvious correlation between the cost to evaluate a query and its predictive power. Therefore it can be expected that cheaper queries result in good features as likely as more expensive ones.

16.4 Related Work and Discussion

A number of approaches for modeling from relational representation have been proposed in the ILP community. Often, these approaches can be described as a "propositionalization," or as an "upgrade" depending on whether feature generation and modeling are integrated. Propositionalization [17] implies separation of modeling from relational feature generation. A logic theory learned with an ILP method can be used to produce binary features for a propositional learner. For example, Srinivasan and King [31] use linear regression to model features constructed from the clauses returned by Progol [21] to build predictive models in a chemical domain. Bernstein et al. [5] introduce a relational vector-space model for classification in domains with linked structure; a derived classifier based on known labels of linked neighbors is proposed in [20]. Decoupling feature construction from modeling, as done in propositionalization, retains the inductive bias of the technique used to construct features; i.e., better models potentially can be built if one allows a propositional learner itself to select its own features based on its own criteria. First-order regression system (FORS) [14] more closely integrates feature construction into regression modeling, but does so using a FOIL-like covering approach for feature construction. Additive, or cumulative, models, such as linear or logistic regression, have different criteria for feature usefulness; integrating feature construction and model selection into a single process is advocated in this context in [6, 25]. Coupling feature generation to model construction can also reduce computational costs by only generating features which will be tested for the model selection.

In contrast to propositionalization, upgrading usually implies that generation of relational features and their modeling are tightly coupled and driven by propositional learner's model selection criteria. SGLR shares these characteristics, and its simpler form, when no cluster-derived relations are used, is, in this sense, an "upgrade" of generalized linear models. TILDE [7] and WARMR [10] upgrade decision trees and association rules, respectively. S-CART [16] upgrades CART, a propositional algorithm for learning classification and regression trees. Dehaspe's MACCENT system [9] uses expected entropy gain from adding binary features to a maximum entropy classifier to direct a beam search over first-order clauses; it determines when to stop adding variables by testing the classifier on a held-out data set. Van Laer and De Raedt present an overview of upgrading in [18]. ILP gives one way to structure the search space; others can be used [27].

16.5 Conclusion

We presented structural generalized linear regression and used its logistic regression variant for analyzing document data from CiteSeer. SGLR combines the strengths of generalized linear regression modeling (e.g., linear, logistic, and Poisson) with the higher expressivity of features automatically generated from relational data sources. New, potentially predictive features and relations in the database are generated lazily, and selected with statistically rigorous criteria derived from the regression model being built. SGLR is applicable to large domains with complex, sparse and noisy data sources; these characteristics suggest focused, dynamic feature generation from rich feature spaces, regression modeling, rigorous feature selection, and the use of query and statistical optimizations, all of which contribute to the expressivity, accuracy, and scalability of SGLR.

SGLR is attractive in offering a factored architecture which allows one to plug in any additive statistical modeling tool and its corresponding feature selection criterion. This contrasts with recursive subdivision methods in which one cannot easily separate out search from modeling and feature selection. The factored architecture offers many advantages, including support for dynamic feature selection.

We showed how clustering can be used to derive new concepts and relations which augment database schema used in the automatic generation of predictive features in statistical relational learning. Clustering improves scalability through dimensionality reduction. More importantly, entities derived from clusters increase the expressivity of feature spaces by creating new first-class concepts which contribute to the creation of new features in more complex ways. For example, in CiteSeer, papers can be clustered based on words giving "topics." Associated with each cluster (or "concept") is a cluster relation (e.g., "on_topic") which then becomes part of more complex feature expressions such as $exists[publishedIn(D1, V), on\_topic(D, C), on\_topic(D1, C)]$. Such richer features result in more accurate models than those built only from the original relational concepts.

We also showed that dynamically deciding which features to generate can lead to the discovery of predictive features with substantially less computation than generating all features in advance, as done, for example, in propositionalization. Native statistical feature selection criteria can give run-time feedback for determining the order in which features are generated. Coupling feature generation to model construction can significantly reduce computational costs. Some ILP systems, such as Progol, also perform dynamic feature generation, albeit with logic models. Many problem domains should benefit from the SGLR or similar methods, including modeling of social networks, bioinformatics, disclosure control in statistical databases, and modeling of other hyperlinked domains, such as the web and databases of patents and legal cases.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Boston, 1995.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of ACM International Conference on Management of Data*, 1998.

[3] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM Journal of Computing*, 8(2):218–246, 1979.

[4] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, 1973.

[5] A. Bernstein, S. Clearwater, and F. Provost. The relational vector-space model and industry classification. In *IJCAI Workshop on Learning Statistical Models from Relational Data*, 2003.

[6] H. Blockeel and L. Dehaspe. Cumulativity as inductive bias. In *Workshop on Data Mining, Decision Support, Meta-Learning and ILP at PKDD*, 2000.

[7] H. Blockeel and L. De Raedt. Top-down induction of logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[8] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, Berlin, 1990.

[9] L. Dehaspe. Maximum entropy modeling with clausal constraints. In *Proceedings of the International Conference on Inductive Logic Programming*, 1997.

[10] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[11] S. Dzeroski and N. Lavrac. An introduction to inductive logic programming. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 48–73. Springer-Verlag, Berlin, 2001.

[12] D. Foster and L. Ungar. A proposal for learning by ontological leaps. In *Proceedings of Snowbird Learning Conference*, Snowbird, UT, 2002.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, 2000.

[14] A. Karalic and I. Bratko. First order regression. *Machine Learning*, 26:147–176, 1997.

[15] L. Kaufman and P. J. Rousseeuw. *Finding Groups In Data: An Introduction to Cluster Analysis*. Wiley-Interscience, Hoboken, NJ, 1990.

[16] S. Kramer and G. Widmer. Inducing classification and regression trees in first order logic. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 140–159. Springer-Verlag, Berlin, 2001.

[17] S. Kramer, N. Lavrac, and P. Flach. Propositionalization approaches to relational data mining. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag, Berlin, 2001.

[18] W. Van Laer and L. De Raedt. How to upgrade propositional learners to first order logic: A case study. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 235–261. Springer-Verlag, Berlin, 2001.

[19] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[20] S. A. Macskassy and F. Provost. A simple relational classifier. In *KDD Workshop on Multi-Relational Data Mining*, 2003.

[21] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[22] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In *Proceedings of ACM International Conference on Principles of Database Systems*, 1998.

[23] C. Perlich and F. Provost. Aggregation-based feature invention and relational concept classes. In *International Conference on Knowledge Discovery and Data Mining*, 2003.

[24] A. Popescul, G. Flake, S. Lawrence, L. H. Ungar, and C. L. Giles. Clustering and identifying temporal trends in document databases. In *Proceedings of the IEEE Advances in Digital Libraries*, 2000.

[25] A. Popescul, L. H. Ungar, S. Lawrence, and D. Pennock. Towards structural logistic regression: Combining relational and statistical learning. In *Proceedings of the Workshop on Multi-Relational Data Mining at KDD-2002*, Edmonton, Canada, 2002.

[26] A. Popescul, L. H. Ungar, S. Lawrence, and D. Pennock. Statistical relational learning for document mining. In *Proceedings of the IEEE International Conference on Data Mining*, 2003.

[27] D. Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

[28] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[29] G. Schwartz. Estimating the dimension of a model. *Annals of Statistics*, 6 (2):461–464, 1978.

[30] E. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

[31] A. Srinivasan and R. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.

[32] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of ACM International Conference on Management of Data*, 1996.

[33] J. Zhou, B. Stine, D. Foster, and L. Ungar. Streaming feature selection using alpha investing. In *International Conference on Knowledge Discovery and Data Mining*, 2005.