Relational Learning for Web Document Classification

Zdravko Markov[1], Ingrid Russell
July, 2007

Overview

Most of the content-based approaches to text and web document classification explored in other related projects are based on the bag of words model, well known from the area of Information Retrieval. This model is simple and efficient, but fails to capture many additional document features such as the internal HTML structure, language structure and inter-document link structure. All this however may be a valuable source of information for the classification task. The basic problem with incorporating this information into the classification algorithm is the need for uniform representation. For example, the content-based classification works well with the vector space representation, while hyperlink-based classification can be implemented by using graph models. This project introduces an approach that allows various kind of information to be represented in a uniform way and used for document classification. The idea is known as *Relational Learning* or *First-Order Learning*. Another term also used in this context is *Inductive Logic Programming* (ILP), which uses the language of logic programming (or Prolog) as a representation language for learning. Some relational learning techniques have been successfully used for Data Mining applications (Relational Data Mining).

The project allows students to study the basics of relational learning and reasoning in the context of solving practical problems. One of the most successful relational learning systems, FOIL is used to create relational representation of web documents and to solve classification problems.

Objectives

The aim of this project is to provide a framework for experimentation and solving practical problems in the area of relational learning and first order inference. By using this framework students can:

1. Learn the basics of Relational Learning and its application to web document classification.
2. Gain experience in using software applications in these areas for solving practical problems.
3. Better understand the fundamentals of First-Order Logic, Learning and Reasoning in First-Order Logic, which are basic components of the wider area of knowledge representation and reasoning in AI.

[1] Corresponding author: markovz@ccsu.edu, Department of Computer Science, Central Connecticut State University, 1615 Stanley Street, New Britain, CT 06050.

Project Description

As in other related projects (see Web Document Classification or Probabilistic Reasoning) we start with the data collection step, where a portion of the web is identified and web documents are collected and organized by topic. At the next step, data preparation we use basic Information Retrieval methods to generate the standard vector representation of our web documents. In contrast to the other web projects however at this step we further explore various relational representations based on document content and also add the information about hyperlinks between documents. In this way we create reach data sets representing a portion of the web based on both content and structure. These sets are then used for relational learning and reasoning in later steps.

Data Collection

To illustrate the process of data collection we use a small part of the Web, a set of pages describing the departments in the school of Arts and Sciences at CCSU. Each of these pages contains a short textual description of the department. Figure 1 shows a hub page that includes links to the department pages. As we are going to classify these and also new documents into categories at this point we have to determine a class label for each department page. These labels can be given independently from the document content however as the classification will be based on the document content some semantic mapping between content and class should be established. For this purpose we can roughly split the departments into two groups – sciences and humanities. Table 1 shows these two groups (the department names are abbreviated).

Documents	**Class**
Anthropology, Biology, Chemistry, Computer, Economics, Geography, Math, Physics, Political, Psychology, Sociology	sciences
Justice, Languages, Music, Philosophy, Communication, English, Art, History, Theatre	humanities

Table 1. Documents groped by class

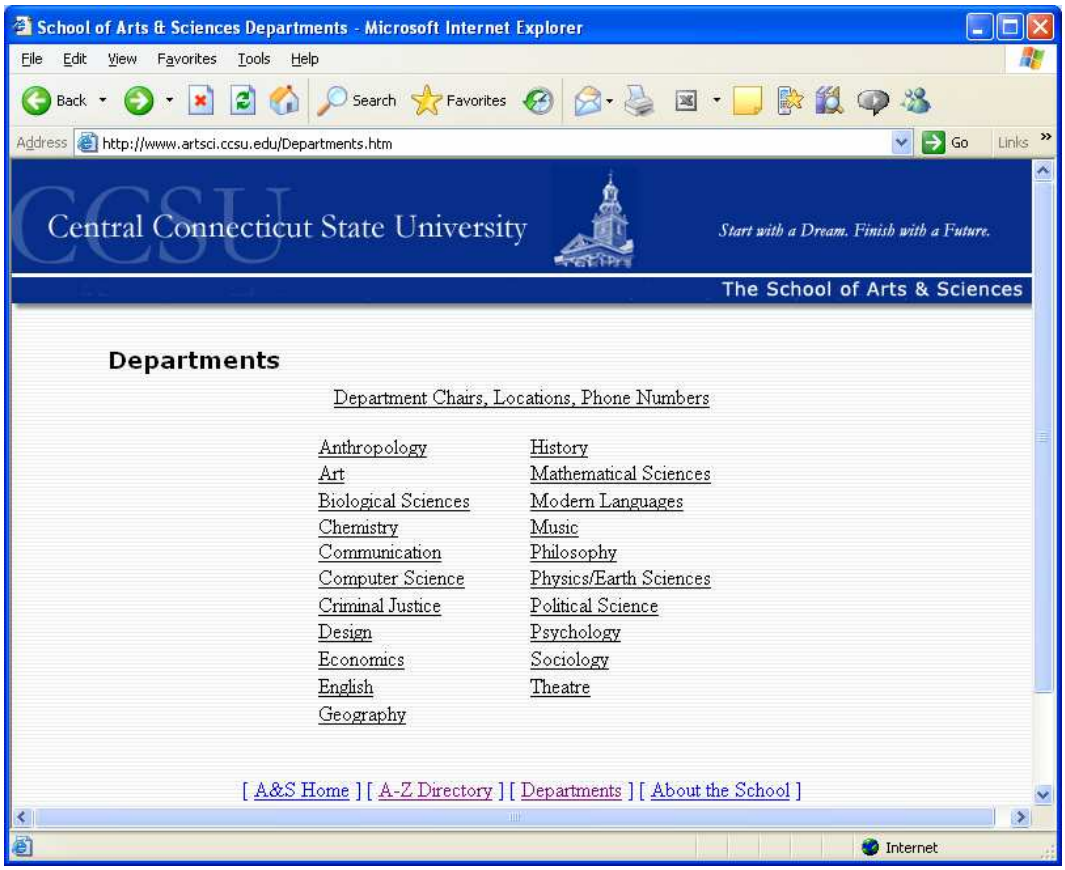

Figure 1. A directory page for a collection of web documents

Further we are going to look into the hyperlink structure of the selected set of pages. For this purpose we may simply follow the links we see inside each page and fill in a table like the one below:

	Departments	Department Directory	Anthropology	Art	…
Departments	0	1	1	1	…
Department Directory	1	0	0	0	…
Anthropology	1	0	0	0	…
Art	1	0	0	0	…
…	…	…	…	…	…

This is the contingency matrix of the portion of the web graph we consider. Each cell is filled with 1 if there is a link from the page in the row to the page in the column, and with 0 otherwise. This matrix is a convenient representation of the web graph when the link-based page rank is computed because it involves matrix calculations. For our purposes however we can also use lists of pages that each page is linked to. For example:

Departments: {Department Directory, Anthropology, Art, …}
Department Directory: {Departments, …}
Anthropology: {Departments, …}
Art: {Departments, …}
…

The link structure can be extracted automatically from the web graph by using a web crawler (see for example, WebSPHINX a free customizable web crawler available from http://www.cs.cmu.edu/~rcm/websphinx/). More about this process can be found in [2], Chapters 1 and 2.

The **deliverable** for the data collection step includes two items:

1. A collection of web documents represented as plain text files grouped into categories (classes). These files will be used to represent the document content. The conversion to text can be done is various ways manually or using some tools. For small number of documents we suggest using the Internet Explorer, which allows the currently loaded HTML document to be saved with the "Save As…" option with "Save as type: Text File (*.txt)". As an illustration we suggest that students look into the collection of text files describing the A&S departments available from the data repository at http://www.cs.ccsu.edu/~markov/dmwdata.zip, folder "CCSU departments".
2. A matrix or list representation of the hyperlink structure of the web graph from where the web documents are collected.

Data preparation

At this step we first apply basic Information Retrieval methods to represent the web documents as vectors. This is the so-called vector space model, where the words in the documents are used as attributes (or dimensions in a metric space) and each document is a vector of values reflecting the occurrences of the corresponding word in that document. This representation is known in Machine Learning as attribute-value representation and is one of the most popular propositional (based on propositional logic) representations. Further we transform this representation in several relational forms, which allow a relational learning system to build various first-order hypotheses for the document class. The relational representation is then further extended with information about the hyperlink structure of our web domain. The result of these transformations is a set of data files to be used by the relational learning system FOIL at the machine learning step.

Vector space representation

At this step we use the Weka data mining system [4] to build the vector space model of the web documents collected during the previous step. More specifically we are going to convert the text documents into data sets in Weka's ARFF format to be further converted into relational form. We suggest that students read Chapter 1 of book [2] (available for free download from Wiley) before continuing with the steps described below:

1. **Download and install the Weka data mining system (version Weka 3.4)** (http://www.cs.waikato.ac.nz/~ml/weka/). Read the documentation and try some examples to familiarize yourself with its use (e.g. load and explore the weather data, provided with the installation).

2. **Create a string data file in ARFF format** (see the description of the ARFF format at http://www.cs.waikato.ac.nz/~ml/weka/arff.html). Follow the directions below:

   First create a **concatenation of all text documents** (**text corpus**) obtained from the data collection step and save them in a single text file, where each document is represented on a separate line in plain text format. For example, this can be done by loading all text files in MS Word and then saving the file in plain text format without line breaks. Other editors may be used for this purpose too. Students with programming experience may want to write a program to automate this step.

   Once the file with the text corpus is created enclose each line in it (an individual document content) in quotation marks (“) and add the **document name** in the beginning of the line and the **document class** at the end, all separated by commas. Also add **a file header** in the beginning of the file followed by @data as shown below:

   @relation departments_string

   @attribute document_name string
   @attribute document_content string
   @attribute document_class string

   @data

   Anthropology, " anthropology anthropology anthropology consists …”, sciences
   …

   This representation uses three attributes – document_name, document_content, and document_class, all of type string. Each row in the data section (after @data) represents one of the initial text documents. Note that the number of attributes and the order in which they are listed in the header should correspond to the comma separated items in the data section. An example of such string data file is “Departments-string.arff”, available from the data repository at http://www.cs.ccsu.edu/~markov/dmwdata.zip, folder “Weka data”.

3. **Create a binary (Boolean) data set in ARFF format**. Load the string data file in Weka using the “Open file” button in “Preprocess” mode. After successful loading the system shows some statistics about the number of attributes (3) their type (string) and the number of instances (rows in the data section or documents).

   Choose the **StringToNominal** filter and apply it (one at a time) to the first attribute, document_name and then to the last attribute (index 3), document_class. Then choose the **StringToWordVector** filter and apply it with onlyAlphabeticTokens=true and useStoplist=true (and the default settings for the other parameters). As Weka moves the class attribute at the second place, move it back last by using the **Copy** filter and the **Remove** button. Finally apply the **NumericToBinary** filter. The result of all these

steps is a Weka data set that uses a binary representation for the documents. An example of this data set is the file "Departments-binary.arff", available from the data repository at http://www.cs.ccsu.edu/~markov/dmwdata.zip, folder "Weka data".

Now we have a **document-term matrix** loaded in Weka. Press the "Edit" button to see it in a tabular format, where you can also change its content or copy it to other applications (e.g. MS Excel). Weka can also show some interesting statistics about the attributes. The class distribution over the values of each attribute (including the document name) is shown at the bottom of the Selected attribute area. With Visualize All we can see the class distribution in all attributes. If we change the class to document_name we can see the distribution of terms over documents as bar diagrams.

4. The current representation of our documents has too many attributes (for Departments-binary.arff it is 612). Many of them are not relevant to the classification task. Also, for our learning experiments we need a small number of relevant attributes, which will be used to build a relational representation. So, we apply some of the Attribute Selection filters to achieve this. For example, the subset evaluator (CfsSubsetEval) select the most relevant subset of attributes, which consists of only 3 attributes (without the class, which is essentially used in the selection process, and thus cannot be eliminated). This is definitely a good subset however we suggest that a little larger subset is selected so that more interesting relational representations are built later. For this purpose we can use another evaluator (e.g. InfoGainAttributeEval) in combination with the Ranker search method to get an ordered list of attributes. Then we select, say, the first ten and remove the rest (keeping the class).

5. Finally we need to save the current data to a text file so that it can be later used to build the relational representation. For this purpose we can use the standard Weka ARFF format. As the default ARFF format is sparse we first convert it to non-sparse by using the SparseToNonSparse filter.

The **deliverable** for the vector space representation step is a non-sparse ARFF data file with a subset of relevant binary attributes.

Relational data models

The document vectors are normally considered as series of attribute values, which is the basic data representation for most machine learning algorithms. Sometimes this representation is called propositional, because each attribute-value pair can be considered as a predicate (statement with a truth value) and the document vector - a conjunction of these predicates. The propositional logic is a subset of the first order logic (or predicate calculus) without variables. So, we may extend our representation to first order logic by considering relations with variables. For example, the vector representing the Anthropology document from our departments domain

```
anthropology,1,1,1,1,1,0,1,0,0
```

can be considered as an instance of the relation `content` with 10 arguments:

```
content(A,B,C,D,E,F,G,H,I,J)
```

where A is a variable taking the values of all document names and each of the rest of variables takes the value of 0 or 1 depending on whether or not the corresponding term (*research, science, concentrations, ba, social, history, biological, copernicus, environmental* in that order) occurs in the document. We may also define a separate relation defining the class of each document:

```
class(A,B),
```

where variable A representing the document name and B – its class (sciences or humanities).

When variables are substituted by their respective values we say that the relation is instantiated. Thus we can get the following instances of the above relations:

```
content(anthropology,1,1,1,1,1,0,1,0,0).
class(anthropology,sciences).
```

In relational learning these instances are considered as input to the learning systems which is supposed to induce a relational definition of the corresponding predicates. For example, in the language of Horn clauses (or Prolog) the following definition may be induced:

```
class(A,sciences) :- content(A,1,D,E,F,G,H,I,J,K).
class(A,humanities) :- content(A,0,D,0,1,0,F,0,G,0).
class(A,sciences) :- content(A,C,1,C,E,C,C,C,F,G).
```

In this definition some variable are instantiated and others are left uninstantiated (free). The latter can take any value. The above clauses are in fact logical implications (rules). For example, the first clause can be read as “if document A includes the term *research* (first variable is instantiated with 1), then its class is *science*. The presence or absence of the rest of the terms does not change the truth value of the implication, so these variables are left uninstantiated.

The presence of free variables in the above clauses suggests another relational representation. The idea is instead of the single relation `content` defining the document content with positionally encoded terms to use an instance of the relation for each term. That is instead of

```
content(anthropology,1,1,1,1,1,0,1,0,0).
```

we use the following set of relations:

```
contains(anthropology,science).
contains(anthropology,research).
contains(anthropology,science).
contains(anthropology,concentrations).
```

```
contains(anthropology,ba).
contains(anthropology,social).
contains(anthropology,biological).
```

This new set of relational data extended with the instances corresponding to the rest of the departments (`art, humanities` etc.) and combined with the instances of the document class may result in the following relational definition:

```
class(A,sciences) :- contains(A,research).
class(A,sciences) :- contains(A,science).
class(A,humanities) :- not contains(A, research),
                       not contains(A, science).
```

In relational learning this definition represents the *target relation*, for which the learning system infers *clauses* (also called *rules*) by using the *background knowledge* - the known instances of all relations (in our examples these are the instances of `content` or `contains`, as well as `class`). The relational language used here is based on Prolog syntax, where ":-" means implication (←), "," – logical conjunction, and "not" – logical negation. Each clause (an instance of a relation also called *fact*, or a rule) should end with a dot (.). Strings beginning with an upper case letter are variables and those beginning with a lower case letter or in single quotes are constants. Thus, these clauses read: "If a document contains "science" its class is A. If a document contains "research" its class is A. If a document does not contain "science" and does not contain "research" its class is B." The last clause uses the so-called *closed world assumption* (CWA) where "not contains(A, science)" means that "contains(A, science)", where A has the value of a particular document, is not present in (or more precisely, does not logically follow from) the background knowledge.

The relational language can be also used to represent the web *hyperlink structure* as well as other relations between web documents. We shall illustrate this with representing a *similarity relation* between web pages.

Let us use the background knowledge consisting of the instances of the `contains` predicate and extend it with two additional sets of instances representing the `link_to` and `similar` predicates.

```
similar(computer,math).
similar(computer,physics).
similar(math,computer).
similar(math,physics).
similar(physics,computer).
. . .

link_to(computer,scholarship).
link_to(math,scholarship).
link_to(physics,scholarship).
```

Note that we now consider not only the text in the web documents, but also HTML links they contain. We also extend the set of web pages with a new page called `scholarship`,

and assume that a link to this page has been included in pages `computer`, `math` and `physics`.

Given the above background knowledge and `similar` as a target predicate, a relational learning system may infer the following set of clauses:

```
similar(A,B) :- link_to(A,C), link_to(B,C), A<>B.
similar(A,B) :- contains(B,social), contains(A,social), A<>B.
```

This definition represents two rules by which we may classify web pages as similar – first, if they share a term (content-based similarity) and second, if they both point to another web page (link-based similarity).

The **deliverable** for the relational data model includes creating three different sets of background knowledge by using the ARFF data from the vector space representation step. The sets should be represented in Prolog syntax (see above).

1. All instances of relation `class` (document name, document class) and the positionally encoded relation `content` (document name and the 9 most relevant attributes).
2. All instances of relation `class` (document name, document class) and two-argument relation `contains` (document name, term). The latter is produced from the 10-argument relation `content` created at the previous set.
3. All instances of relation `contains` (from the previous step) extended with two additional sets of instances representing the `link_to` and `similar` relations. Use additional web pages with links from the original set of department pages (as the `scholarship` page used in the example above).

Relational Learning

There are various algorithms for relational or First-Order learning. Hereafter we shall briefly describe the algorithm FOIL, which is simple and efficient and uses an entropy-based technique to guide the search for clauses similar to the one used in the popular propositional learning system ID3 (learning decision trees).

Let us look into the process of creating the clauses for the `class` relation. The instances of this relation which are a part of the background knowledge are called *positive examples*. Given the positive examples the algorithm automatically generates a set of *negative examples* by using the CWA approach (there is also an option that allows specifying them explicitly). The negative examples are instances of the same relation, however with class values taken from the contrasting class. That is, if the document belongs to sciences, then the negative example includes humanities and vice versa. The objective of the learning system is to find clauses that *cover*[2] all positives and do not

[2] A clause covers an example if there are substitutions for its variables, such that when applied the clause head coincides with the example and each of its body literals occurs in (or follows from) the background knowledge.

cover any negatives (the latter condition may be relaxed in the presence of noise, i.e. some percentage of negative examples may be covered too).

The algorithm works in a general to specific fashion starting from the most general hypotheses, “class(A,sciences)” and “class(A,humanities)”. Because “A” is free variable, this clauses cover all positives, however all negatives too. Therefore they should be specialized by adding body literals. The candidates are generated using relations and arguments from the background knowledge. Also, they have to include the variable “A”, because otherwise the possible values “A” are not restricted and thus the coverage does not change. For example, a clause like “class(A, science) :- contains(X,history)” is equivalent to “class(A,science)”. The potential candidate literals along with the number of positive and negative examples covered after adding the literal are shown in the table below for one of the target clauses, “class(A,sciences)”.

Candidate literal	**# of covered positives**	**# of covered negatives**
contains(A, history).	0	3
contains(A, science).	7	0
contains(A, research).	8	0
contains(A, offers).	3	6
contains(A, students).	10	5
contains(A, hall).	9	6

After generating all candidates the algorithm computes an *information gain* evaluation function, which selects the best literal. We are not going into the details, but the idea is to maximally reduce the total number of bits needed to encode the classification of all positive examples covered by the clause. In the particular case, “contains(A, research)” and “contains(A, science)” are the clear winners (no negatives covered), which comes at no surprise because the attributes “research” and “science” are also on top of the entropy ranking that may be used in the Vector Space representation step for finding relevant attributes. Thus the algorithm picks “contains(A, research)” and creates the clause:

```
class(A,sciences) :- contains(A,research).
```

This clause does not cover any negative examples. So, it does not need any further specialization and the algorithm stops looking for more literals to add. Otherwise, the clause would have to be extended with more literals until it covers no negative examples. This terminates the inner loop of the algorithm, which at each run creates a single clause.

If the clause found by the inner loop covered all positive examples, the algorithm would stop. However it covers only 8. Therefore more clauses are needed to cover the rest of the positive examples. The 8 already covered examples are excluded from the original set of positives and the algorithm enters the inner loop again with the remaining ones: {class(chemistry, sciences), class(computer, sciences), class(geography, sciences)}. Now it finds the clause “class(A, sciences) :- contains(A, science)”, which happens to cover all remaining examples.

The clause for class(A,humanities) is inferred in a similar way. Note that in this case the algorithm picked negated literal to add to the clause body, like “not contains(A, science)”, which obviously showed the best information gain score.

The FOIL system is available for free from the web page of its author Ross Quinlan at http://www.rulequest.com/Personal/. Appendix 1, “Installing and Running FOIL on Windows” provides instructions for installing, running and getting familiar with the system.

Let us now apply FOIL to learn the definition of “class” using the background knowledge with the “contains” relation. First we have to prepare the data file. We use the data in Prolog syntax from the previous step. For example, the set of facts for relation “class” is the following:

```
class(anthropology,sciences).
class(art,humanities).
class(biology,sciences).
class(chemistry,sciences).
class(communication,humanities).
class(computer,sciences).
class(justice,humanities).
class(economics,sciences).
class(english,humanities).
class(geography,sciences).
class(history,humanities).
class(math,sciences).
class(languages,humanities).
class(music,humanities).
class(philosophy,humanities).
class(physics,sciences).
class(political,sciences).
class(psychology,sciences).
class(sociology,sciences).
class(theatre,humanities).
```

Next we determine the *type* of each argument of the relations. In our example the relations use the so called *discrete types*, which are specified by providing the set of all values (or constants in FOIL’s terminology) a particular argument of a relation may take. For example, the first argument of relation “class” is a document name and its type specification following the FOIL syntax is:

```
document_name: anthropology, art, biology, chemistry, communication,
computer, justice, economics, english, geography, history, math,
languages, music, philosophy, physics, political, psychology,
sociology, theatre.
```

The type specification of the second argument of “class” is:

```
document_class: *sciences,*humanities.
```

The constants starting with * may appear in the learned definition of the target relation (as for example, in “class(A, sciences) :- contains(A, research)”). The type name is user defined (we choose document_name and document_class) and appears in the definitions of the relations that follow the type specifications. For the class relation this is (note that the dot after the set of facts if part of the FOIL syntax):

```
class(document_name,document_class)
anthropology,sciences
art,humanities
biology,sciences
chemistry,sciences
communication,humanities
computer,sciences
justice,humanities
economics,sciences
english,humanities
geography,sciences
history,humanities
math,sciences
languages,humanities
music,humanities
philosophy,humanities
physics,sciences
political,sciences
psychology,sciences
sociology,sciences
theatre,humanities
.
```

The header of all relations other than target relations begins with *. Thus the “contains” relation is defined similarly, however we add * in the beginning to indicate that this is not the target relation.

```
*contains(document_name,term)
```

Here we also need to define the specification of type “term”, which in our example includes the 9 terms selected earlier (listed with * in the beginning as we want them to appear in the target definition).

After we include all type specifications and relation definitions in a file we supply it as input to FOIL, which then generates a definition for the target relation “class”. Note that this definition may vary depending on the parameter setting used to run FOIL (e.g. –n or –N options to control the use of negation) as well on the particular choice of terms used to form the definition of the “contains” relation.

The **deliverable** for this step includes FOIL data files and results from running the system (definitions of target relations) for each of the three sets of background knowledge created at the “Relational Data Models” step.

Relational Reasoning

After being generated the clauses for the target relation can be used to infer various types of information from the background knowledge and also to classify new documents. For this purpose we need a deductive reasoning system that works with relational definitions such as Prolog. For this project we use one of the most popular Prolog versions, SWI-Prolog, which is available for free from http://www.swi-prolog.org/.

After installing and getting familiar with SWI-Prolog (see Appendix 2: Installing and running Prolog) we prepare the datasets for our experiments. We use the data files created at the "Relational data models" step as they are in Prolog syntax and then combine their content with the relational definitions generated by FOIL at the "Relational learning" step. The basic idea here is to replace the set of instances of the target relation, which was used by FOIL with the set of clauses that FOIL generated for the target relation. It is important to note that Prolog does not require any type definitions, so we skip the type specifications needed by FOIL. For example, we combine the instances of "contains" with the definition of "class" generated by FOIL (excluding the instances of "class" used by FOIL) and get the following Prolog data set (note that the list of instances of "contains" shown below is incomplete):

```
contains(anthropology,research).
contains(anthropology,science).
contains(anthropology,social).
contains(anthropology,ba).
contains(anthropology,concentrations).
contains(anthropology,students).
contains(art,ba).
contains(art,students).
…

class(A,sciences) :- contains(A,research).
class(A,sciences) :- contains(A,science).
class(A,humanities) :- not contains(A, research),
                       not contains(A, science).
```

Now we are ready to use Prolog for relational reasoning. First we load this file in Prolog (through file>consult, or by typing ['filename.pl'] at the prompt). Then we can type various queries. For example, to find the class of document "computer" we type

```
?- class(computer,X).

X = sciences

Yes
?-
```

Or for "art" we get

```
?- class(art,X).
```

```
X = humanities

Yes
?-
```

The Prolog answer is X=sciences. Prolog also provides alternative answers (by entering ";" after the first answer). For example,

```
?- class(X,sciences).

X = anthropology ;

X = biology ;

X = economics ;

X = math

?-
```

Appendix 2 lists some sources that provide more information about using Prolog.

Assume now that we extract the terms from an unknown document and generate the corresponding set of instances of the "contains" relation. Then we add the definition of "class" (the one generated by FOIL from a set of documents with known classes) and obtain a dataset that can be used to infer the class of the unknown document. This is an example of classification based on relational data models.

Further we can use the data sets that include information about the web page link structure combined with our definition of "similar" (learned by FOIL) to find similar web pages. Below are examples of Prolog queries for this purpose.

```
?- similar(computer, art).

No
?- similar(computer, math).

Yes
?- similar(computer, X).

X = math;

X = physics

?-
```

We may also change the definition of similar generated by FOIL to a more general one:

```
similar(A,B) :- link_to(A,C), link_to(B,C), A<>B.
similar(A,B) :- contains(B,C), contains(A,C), A<>B.
```

The second clause here expresses the idea that two pages are similar if they share a term. With this definition we may get a richer set of similar pages. This new definition may be also learned by FOIL with a proper data set and parameter setting.

The **deliverable** for this step includes the preparation of Prolog datasets that include all target relations created by FOIL at the relational learning step ("class" and "similar") and results from various queries with these data. **Advanced projects** may include preparation of datasets from new documents or web pages and using Prolog for determining document class, similarity between unknown documents or other relationships defined by their instances and learned as target relations by FOIL.

Prerequisites and requirements

Students should have basic knowledge of discrete mathematics and logic. The required background for this project includes reading Chapters 13 and 14 of Russell and Norvig's book [1] and Chapter 6 of Tom Mitchell's ML book [3]. These two sources provide the basic knowledge needed for the relational learning and reasoning step. Some programming experience in Prolog would be helpful as this language is a practical implementation of First-Order logic and provides the basic inference tools needed at the relational reasoning step. The data collection and data preparation steps require basic knowledge of Web Mining and Information Retrieval, which is provided in Chapter 1 of Markov and Larose's book [2]. Practical information needed to use the Weka system, the tool used in these steps, is provided in Witten and Frank's book [4].

References

1. Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, Second Edition, Prentice Hall, 2003. Chapters 13, 14.
2. Zdravko Markov and Daniel T. Larose. *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*, Wiley, 2007. Chapter 1 is available for free from http://media.wiley.com/product_data/excerpt/56/04716665/0471666556.pdf
3. Tom Mitchell. Machine Learning, McGraw Hill, 1997, Chapter 6.
4. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques* (Second Edition), Morgan Kaufmann, 2005.

Appendix 1: Installing and running FOIL on Windows

- Download the shell archive from http://www.rulequest.com/Personal/foil6.sh.
- Install Cygwin (Linux-like environment for Windows) from http://www.cygwin.com/ (use the installer http://www.cygwin.com/setup.exe). For compiling FOIL you need gcc, gdb, and make (available from the Devel package). You also need an editor from the Editors package (e.g. vi).

- Run Cygwin and type ‘sh foil6.sh’ in the folder where foil6.sh is located. This extracts from the shell archive the FOIL source files, examples, a manual and a readme file.
- In Cygwin type ‘make’. This compiles FOIL and creates and executable file foil6.exe.
- Run the first example to learn the definition of the member predicate:
  ./foil6 -n <member.d (the output goes to the console) or
  ./foil6 -n <member.d > member.out (the output goes to the file member.out)
- Read member.explain to understand the format of the input/output and FOIL parameters.
- Read the manual (MANUAL.txt) provided with the installation. It explains the data format and the different options and settings available when running FOIL.
- Always use an editor in Cygwin (e.g. vi) to create and edit data files for FOIL input. The Windows editors (Notepad, WordPad, Word) use a different delimiter characters (CR/LF), which FOIL does not accept. You may use copy/paste to move large blocks of data between Windows and Cygwin (use the command prompt edit options and turn “vi” in insert mode).
- When preparing the data files strictly follow the format explained in the manual and use the examples provided with the installation. Note that white spaces and blank lines are part of the format and have to be used accordingly. For example, no blank lines are allowed between the definitions of the relations.

Appendix 2: Installing and running Prolog

- Download SWI-Prolog from http://www.swi-prolog.org/. Use the stable versions and the self-installing executable for Windows. For this project you need only the basic components, so you may uncheck all optional components.
- Read Quick Introduction to Prolog (http://www.cs.ccsu.edu/~markov/ccsu_courses/prolog.txt) and do all experiments described in it.
- There also more detailed Prolog tutorials:
  - A Prolog Tutorial by J.R. Fisher (http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html)
  - More tutorials: http://www.swi-prolog.org/www.html