Capita Selecta AI

Statistical Relational Learning

October 19, 2015

- Preferably work in teams of two students. It is not allowed to share answers between teams.
- Send a report containing your answers and explanations (in pdf) and the ProbLog input files for your models to wannes.meert@cs.kuleuven.be **before 6pm on Friday January 8, 2015** (one submission per team, with the second team member in cc).
- Grades for this part are fully determined by the submission (no personal defense / presentation).
- You can use the ProbLog online interface at `https://dtai.cs.kuleuven.be/problog` to experiment with your models. However, it has a time limit that is too low for fully answering some of the questions. We therefore recommend using the offline version available from[1] `https://dtai.cs.kuleuven.be/problog/index.html#download`.

1 Probabilistic Databases (5 points)

1.1 Database

Construct your own *probabilistic* database.[2] Make sure it has at least

- three tables/relations,
- one binary relation (a relation with two arguments), and
- ten tuples/rows.

Show the database in tabular form in your report. Also show the equivalent ProbLog program.

1.2 Query

For your own probabilistic database, construct two example queries. First, construct a non-Boolean query and write it in three languages

[1] Documentation is available on `http://problog.readthedocs.org`

[2] If you lack inspiration, you can model the relationships in your favorite TV show.

- SQL,
- First-order logic (as a FO formula), and
- ProbLog.

Second, construct a Boolean query and write it in two languages

- First-order logic (as a FO sentence), and
- ProbLog.

Make sure that the queries involve at least two tables, and that answering them requires looking at minimum 5 tuples.

1.3 Inference

For the Boolean query you have chosen above, indicate how you can compute its probability manually. Write the answer using simple arithmetic (product, sum, and subtraction) and the probabilities in the database. Also compute the probability with ProbLog to verify the result.
Optional: How do the lifted inference rules answer this query?

2 Beverage Preferences (7 points)

2.1 Preferences

Consider a data set in which we collected the drink preferences of a number of people. This could be the result of, for example, a questionnaire or an inspection of a social network. For a certain number of beverages the table contains information about whether a person 'not likes it', 'likes it a little', 'likes it much', 'likes it very much'. The results can be summarized in a table as follows:

```
preference(john,  whiskey,   like_little).
preference(john,  gin,       like_little).
preference(john,  lager,     like_much).
preference(mary,  whiskey,   like_verymuch).
preference(mary,  redwine,   like_much).
preference(peter, whiskey,   like_much).
preference(peter, stout,     like_much).
preference(peter, whitewine, like_no).
preference(paul,  whiskey,   like_no).
preference(paul,  coffee,    like_verymuch).
preference(ann,   gin,       like_verymuch).
preference(ann,   lager,     like_little).
```

Your task is to infer who will be interested in buying a bottle of whiskey. Create a ProbLog file that answers the following question:

```
query(buy(Person, whiskey)).
```

Using the information that someone who does not like a drink will not buy it, who likes it a little buys it with probability 0.1, who likes it much with 0.5 and who likes it very much with 0.9.

2.2 Subjective Information

The problem with the above information is that the degree of 'liking' something is a vague concept. Even if we would directly question these people some people will say that they do not like a drink while others might express the same preferences as as liking it only a little. In other words, there is a potential overlap between the preferences.

Model a `preference_soft(Person, Drink, Like)` relation that takes into account this subjectivity. Concretely, if we know that someone 'not likes' a drink this can be considered to be 'like a little' with a probability of 0.5 and the other way around. All combinations can be summarized as:

```
P(like_no = like_little) = 0.5
P(like_no = like_much) = 0.1
P(like_no = like_verymuch) = 0.0
P(like_little = like_much) = 0.4
P(like_little = like_verymuch) = 0.1
P(like_much = like_verymuch) = 0.4
```

Rewrite your `buy` predicate to take into account `preference_soft` instead of `preference` directly. The probability of `preference(person, drink, like)` should be the same as `preference_soft(person, drink, like)` for every person-drink-like tuple that is present in the table above. Is this the case?

2.3 Beverage Ontology

Suppose now we want to infer who will buy a new drink that is not yet in our database of preferences. To achieve this we can use background knowledge about beverages. The background knowledge can be, for example, a knowledge graph or ontology that expresses which terms are related, synonyms, ...[3]. For this exercise, you have encode the small ontology fragment depicted in Figure 1 in your ProbLog model.

In this exercise we will assume that a person likes drinks that are in a category that is close to categories that contain drinks that she likes. Alter `preference_soft` such that a preference propagates to another drink with a probability of $\frac{1.0}{l}$ with l the number of edges you traverse to connect two drinks. For example, if we know that someone likes a Stout a little then she likes an Ale a little with probability $\frac{1.0}{2} = 0.5$ and a white wine with probability $\frac{1.0}{4} = 0.25$.

Using this model, can you predict who is most likely to buy an ale, a drink that is not in our database.

What other sources of background information do you think are useful to add to the model?

3 Document Classification (8 points)

Probabilistic programming languages such as ProbLog allow for easy prototyping of probabilistic models for a variety of applications. We will build a model to predict the topic of a document based on the words appearing in that document.

[3] A popular knowledge graph is WordNet `http://wordnetweb.princeton.edu`

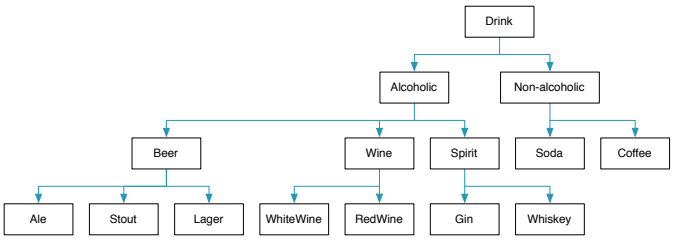

Figure 1: Fragment of a beverage ontology

3.1 Pre-processing

The first step, the pre-processing has been performed already for you and the resulting files are available together with this assignment. About 100 Reuters news articles from 1987 are selected with either the topic 'interest' or 'grain'[4]. All the words appearing in those articles are stemmed, i.e. reduced to its root form, and stop words are removed. Furthermore, the top 10, 20, 30, 40 and 50 words according to TF-IDF are selected and can be found as Prolog facts in directories `data_x0`. In `reuters_ev.pl` all the evidence facts are listed that describe the training data. Similarly, a separate set of test facts is prepared in `reuters_facts_text.pl`. If you want to know how the data is prepared or would like to use your own custom set of documents, a makefile and Python script are included for you to execute or inspect.

3.2 Noisy-OR Classifier

The default combination rule of ProbLog rules is a noisy-OR interaction. This makes the noisy-OR classifier a simple model to express in ProbLog [5]. A simple noisy-OR classifier with a Boolean class and two features can be expressed as follows:

```
0.4::weight(1).
class :- feature(1), weight(1).
0.3::weight(2).
class :- feature(2), weight(2).
```

You should make one classifier per topic (thus two) that expresses whether the document has that class (`class=true`) or not (`class=false`). To learn the weights from examples (partial interpretations), the weights can be replaced with `t(_)` and saved in a file, say `reuters_model.pl`:

```
t(_)::weight(1).
class :- feature(1), weight(1).
t(_)::weight(2).
class :- feature(2), weight(2).
```

ProbLog's LFI learning algorithm can now be called as follows:

[4] `http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`
[5] Optional: J. Vomlel. Noisy-or classifier. Int. Journal of Intelligent Systems 21.3 (2006)

```
$ ./problog-cli.py lfi -O reuters_model_trained.pl -v
                    reuters_model.pl reuters_ev.pl
```

Remember that LFI learns from *partial* interpretations meaning that missing examples are not considered to be false.

To test the unseen data you concatenate the trained model file and the test set facts. The resulting file is processed by ProbLog and the resulting class probabilities are printed:

```
$ cat reuters_model_trained.pl reuters_facts_test.pl > reuters_test.pl
$ ./problog-cli.py reuters_test.pl
```

Your task is to build a noisy-OR classifier using the predicates available in the data files. Replace the rules with the `something` predicate with more useful rules. What is your observation for the different sizes of data sets (accuracy, time)? What is the best number of features? 10, 20, 30, 40 or 50 words? Note also that the supplied files summarize all examples as one large interpretation to make it easy to predict multiple unseen documents at once. What would happen if you split this into one interpretation per document?

3.3 Background Knowledge

The naive noisy-OR model is probably too simple for a real world model. When looking at the words that are used in the model, how would you add additional background knowledge to your classifier? For example, you can make the model hierarchical by clustering concepts like all the grains. Can you also relate this to the idea of ontology that you implemented in the previous exercise? Try to add one bit of background knowledge. How does this influence your classification?

3.4 Alternative Classifiers

Traditionally a more popular classifier is the Naive Bayes model. Alter the model you have build in 3.2 to express a Naive Bayes model. The simple example above can be written as a Naive Bayes model as follows:

```
t(_)::class.
t(_)::weight(1,t).
t(_)::weight(1,f).
feature(1) :-   class, weight(1,t).
feature(1) :- \+class, weight(1,f).
t(_)::weight(2,t).
t(_)::weight(2,f).
feature(2) :-   class, weight(2,t).
feature(2) :- \+class, weight(2,f).
```

The Naive Bayes model is popular because the features are independent given the class which leads to a small number of parameters and fast inference. Does this model increase the accuracy of your topic prediction? Can you apply background knowledge to this model? What would be the impact on the efficiency of reasoning on this model.