Lab session

Uncertain data management

Antoine Amarilli and Silviu Maniu

January 23rd, 2017

The goal of this lab session is to play with relational formalisms to manipulate probabilistic data. This is done with MayBMS[1], a probabilistic database implementation built on top of PostgreSQL[2].

MayBMS is ready to be used on your machines with the standard PostgreSQL command line client. To execute MayBMS, open a session on the machine in front of you, open a terminal emulator, and execute the following (`~amarilli`, with a tilde sign, refers to the home folder of user `amarilli`):

```
~amarilli/maybms.sh
```

The initialization will create a database and start the MayBMS server, which may take a few minutes. Once the process completes, you should have a PostgreSQL client ready (ignore the error reported about "application_name"), which you can use for the questions in the lab session that require it. Once you terminate the session, the server will close. You can re-run the same script after the session has closed to start the server again and start a new session, while keeping your existing database.

We consider a *truth finding* application, where we have extracted facts from several sources (i.e., websites). A first table, Trust, indicates which sources are trustworthy. Another table, Claims, indicates which sources support which fact. Both tables are uncertain: we do not know which sources are correct, and we are not sure of whether a source supports a statement (i.e., errors may have been made when extracting facts from sources).

We represent the uncertainty on these two relations using the TID model. We consider the following instance:

Trust

source	proba
Legifrance	0.95
Wikipedia	0.8
Doctissimo	0.4
Gorafi	0.2
Onion	0.1

Claims

source	claim	proba
Legifrance	42	0.7
Wikipedia	42	0.9
Gorafi	42	0.6
Wikipedia	51	0.7
Doctissimo	51	0.4
Wikipedia	66	0.3
Gorafi	66	0.9

[1] `http://maybms.sourceforge.net/`
[2] `https://www.postgresql.org/`

1 Query evaluation by hand

Question 1. We wish to determine the answer to the following query: "Which sources in our dataset are trustworthy and make at least one claim?"

Write this query in the relational calculus. Write the query in the relational algebra using the select, rename, product and project operators.

Write the query again in the relational algebra using the project and join operators, in two equivalent ways (each way should use exactly one project and one join).

Question 2. Consider the deterministic instance obtained from the given TID instance by removing the proba column. What is the result of evaluating the previous query on this deterministic instance?

Question 3. Compute the probability according to the given TID instance that the source Gorafi makes at least one claim. Deduce the probability that Gorafi is a trustworthy source that makes at least one claim.

Hint: Use a calculator or the computer for numerical computations.

Question 4. Generalizing this process, compute the answer to the query of Question 1 with the correct probabilities according to the TID model. The result should be a probabilistic annotation of the result of Question 2. Do this in two steps: first, compute for each source the probability that it makes a claim, and then combine this with the Trust table to obtain the final answer.

Question 5. Consider the two relational algebra queries of Question 1 written with the join and project operators. Extending the relational algebra operators to manipulate probabilities as seen in class, which query would yield the output table of Question 4 with the correct probabilities? Explain why.

Would the other query give the correct output tuples (up to the probabilities)? How would the probabilities compare to the correct output? Explain why.

2 Query evaluation using PostgreSQL

In this section, we will check the result of the previous section using the ordinary SQL features of PostgreSQL, without using MayBMS's specific features.

Question 1. Create the relational tables corresponding to the instance of Trust and Claims presented before, and insert the values (including the probabilities). The types to use are `varchar`, `float`, and `int`.

Question 2. Write an SQL query that computes the output to the query of the previous section, ignoring the probabilities.

Question 3. Inspired by the relational algebra query of the previous section, we will write an SQL query that projects the Claims table to the **source** column and computes the correct probabilities.

For each tuple t in this independent projection, calling $p_1, \ldots, p_n$ the probability of the tuples of Claims that project to t, what is the probability that t is in the projection, as a function of $p_1, \ldots, p_n$?

Following this answer, using the SQL `GROUP BY` operator, write a query that computes the projection of the Claims table to **source**, with the correct probabilities, and store it in a new table T1.

Hint: As there is no `PRODUCT` aggregate function in SQL, use the `SUM` aggregate and the `exp` and `ln` functions to write the query. Alternatively, if you prefer, you may define custom functions and aggregates for the task[3].

Check that the probabilities that you obtain in the table T1 correspond to the intermediate result computed by hand for Question 4 (before using table Trust) in the previous section.

Question 4. Write a query that joins the table T1 with the table Trust and computes the final query result with the correct probabilities in column **proba**. Store this result as the intermediate table T2. Check that you obtain the same result as in the end of Question 4 in the previous section.

3 Query evaluation using MayBMS

We consider again the truth finding application of the previous sections. We now want to execute the same queries as before, but this time using the specific features of the probabilistic DBMS that we are using, MayBMS[4]. We will use the Trust and Claims table created in the previous section.

Question 1. Create tables TrustP and ClaimsP that are the probabilistic counterparts of the deterministic tables Trust and Claims, using the MayBMS `PICK TUPLES` command.

Question 2. We will now use MayBMS to evaluate the query studied in the previous sections. Contrary to those sections, we will be able to leave probability evaluation to MayBMS, in particular we need not worry about safe plans.

Create a table T3 containing the result of evaluating the query (you should be able to use a query that is similar to that of Question 2 in the previous section). Examine the output and check that you understand the additional columns.

Question 3. To obtain the final answer, compute the probability of each source using the `conf()` aggregate. Check that the results are the same as before.

Question 4. Let us now showcase the power of MayBMS with a more complex query: "Which claims are stated by at least two different sources which are both trustworthy?". Write this query in the relational algebra, translate it to an SQL query, and execute it on the TrustP and ClaimsP tables. Use this to compute the probability, for each claim, that it was stated by two different trustworthy sources. Check this answer against the original tables.

4 Hard queries and approximation

In this section, we will study a different query with MayBMS.

[3] See `https://www.postgresql.org/docs/8.3/static/xaggr.html` for an intuitive explanation and see the reference `https://www.postgresql.org/docs/8.3/static/sql-createaggregate.html` and `https://www.postgresql.org/docs/8.3/static/sql-createfunction.html`

[4] A manual and quick tutorial can be found at `http://maybms.sourceforge.net/manual/index.html`

Question 1. Create the following additional table in SQL, which indicates uncertainty about whether a claim is relevant to the user:

Relevant	
claim	**proba**
42	0.2
51	0.4
66	0.8

Question 2. We wish to determine whether there is a trustworthy source that makes a relevant claim. Ignoring the probability, write this query in the relational calculus, in the relational algebra, in SQL, and evaluate it. Is the query true on the input instance?

Question 3. We now wish to determine the probability of this query using MayBMS. Using the two previous probabilistic tables TrustP and ClaimsP, and creating the probabilistic table RelevantP, evaluate the query and compute the probability that it is true.

Question 4. We will now generate a larger synthetic dataset to evaluate the same query on larger data, in more realistic conditions. While PostgreSQL is running, open a second terminal, run the following generation script, and observe its output:

```
~amarilli/gen.py
```

Now, while PostgreSQL is running, execute the SQL commands produced by the script, by passing them to the PostgreSQL client. To do so, run the following in your second terminal:

```
~amarilli/gen.py | psql -h localhost
```

Once this has completed, returning to your PostgreSQL session, check that the tables Trust2, Claims2 and Relevant2 were successfully created.

Evaluate the same non-probabilistic query as in Question 2 on these tables.

Question 5. Create probabilistic tables Trust2P, Claims2P, and Relevant2P from the new tables, in the same way as before. Run the same query as in Question 3 to create a probabilistic table Res2 containing the query output. Now, compute the probability that Res2 contains a tuple.

What happens? Why? You can use the `top` or `htop` command in a separate terminal to check what is your machine doing (press `q` to quit).

To terminate the execution, you can issue the following in a separate terminal:

```
pkill psql
pkill postgres
pkill -9 psql
pkill -9 postgres
```

and restart the `~amarilli/maybms.sh` script.

Question 6. Use the `aconf` operator on Res2 to *estimate* the probability that the query is true. Does running the `aconf` query multiple times result in the same output? Experiment with various values of the `aconf` parameters to see how the running time and result variability is affected.