Lab session

Uncertain data management

Antoine Amarilli and Silviu Maniu

December 19th, 2017

The goal of this lab session is to play with relational formalisms to manipulate probabilistic data. The first part of the assignment asks you to perform some tasks by hand; the second asks you to perform them with PostgreSQL[1]; the rest of the assignment uses MayBMS[2], a probabilistic database extension of PostgreSQL.

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 (each fact is given as a numeric identifier). 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 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 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).

Answer. In the relational algebra, with select, rename, product, and project:

$$\pi_{\mathbf{source}}(\sigma_{\mathbf{source}=\mathbf{source2}}(\mathsf{Trust} \times \rho_{\mathbf{source}\to\mathbf{source2}}(\mathsf{Claims})))$$

[1] `https://www.postgresql.org/`

[2] `http://maybms.sourceforge.net/`

In the relational algebra, with project and join (first possibility):

$$\pi_{\mathbf{source}}(\mathsf{Trust} \bowtie \mathsf{Claims})$$

In the relational algebra, with project and join (second possibility):

$$\mathsf{Trust} \bowtie \pi_{\mathbf{source}}(\mathsf{Claims})$$

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?

Answer. The query result is:

$\mathsf{Trust} \bowtie \pi_{\mathbf{source}}(\mathsf{Claims})$
source
Legifrance
Wikipedia
Doctissimo
Gorafi

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.

Answer. As the two corresponding facts in Claims are independent, the probability that Gorafi makes at least one claim is:

$$1 - (1 - .6) \times (1 - .9) = .96$$

As this is independent from the Trust fact indicating whether or not Gorafi is trustworthy, the probability that Gorafi is trustworthy and makes at least one claim is:

$$.96 \times .2 = .192$$

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.

Answer. We first compute the intermediate result:

$\pi_{\textbf{source}}(\mathsf{Claims})$

source	**proba**
Legifrance	.7
Wikipedia	.979
Doctissimo	.4
Gorafi	.96

And then the final result:

$\mathsf{Trust} \bowtie \pi_{\textbf{source}}(\mathsf{Claims})$

source	**proba**
Legifrance	.665
Wikipedia	.7832
Doctissimo	.16
Gorafi	.192

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.

Answer. The evaluation that we performed corresponds to the relational algebra query:

$$\mathsf{Trust} \bowtie \pi_{\textbf{source}}(\mathsf{Claims})$$

The reason why this way of writing the query (i.e., query plan) yields the correct probability is because this is a *safe plan*.

The other plan would yield the correct output tuples, because the two relational algebra queries are independent (as relational algebra queries, without probabilities). However, the probabilities are incorrect. Indeed, focusing on the output tuple "Gorafi", the two relevant tuples in the intermediate result would be:

$\mathsf{Trust} \bowtie \mathsf{Claims}$

source	**claim**	**proba**
Gorafi	42	.12
Gorafi	66	.18
⋮	⋮	

The result of projecting this would be:

$\mathsf{Trust} \bowtie \mathsf{Claims}$

source	**proba**
Gorafi	.2784
⋮	⋮

Hence, the probability given to the output tuple "Gorafi" is incorrect. However, it is an upper bound of the correct probability (observe that .2784 > .129). The same is true for all tuples, as seen in class.

2 Query evaluation using PostgreSQL

In this section, we will check the result of the previous section using the ordinary SQL features of PostgreSQL. We will do so using the PostgreSQL installation that comes with MayBMS, but we will not be using any MayBMS-specific features.

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`, where the script is stored):

```
~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.

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`.

Answer.

```
CREATE TABLE Trust(source varchar, proba float);
INSERT INTO Trust(source, proba) VALUES
    ('Legifrance', .95), ('Wikipedia', .8), ('Doctissimo', .4),
    ('Gorafi', .2), ('Onion', .1);
CREATE TABLE Claims(source varchar, claim int, proba float);
INSERT INTO Claims(source, claim, proba) VALUES
    ('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);
```

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

Answer.

```
SELECT DISTINCT T.source FROM Trust T, Claims C WHERE T.source = C.source;
```

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].

Answer.

```
CREATE TABLE T1 AS
SELECT source, 1 - exp(sum(ln(1 - proba))) AS proba FROM Claims
    GROUP BY source;
```

Alternatively, the same can be done using custom aggregators to avoid the hack with `exp` and `ln`:

```
CREATE FUNCTION compl(float) RETURNS float AS $$SELECT 1 - $1$$ LANGUAGE SQL;
CREATE FUNCTION disj_or1(float, float) RETURNS float AS
    $$SELECT $1 * (1 - $2)$$ LANGUAGE SQL;
CREATE AGGREGATE disjoint_or(BASETYPE=float, SFUNC=disj_or1, STYPE=float,
    FINALFUNC=compl, INITCOND=1);
CREATE TABLE T1 AS
SELECT source, disjoint_or(proba) AS proba FROM Claims GROUP BY source;
```

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.

Answer.

```
CREATE TABLE T2 AS
SELECT T.source, T.proba * T1.proba AS proba FROM Trust T, T1
    WHERE T.source = T1.source;
```

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.

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

Answer.

```
CREATE TABLE TrustP AS PICK TUPLES FROM Trust
    INDEPENDENTLY WITH PROBABILITY proba;
SELECT * FROM TrustP;
```

This outputs the following table:

```
   source    | proba | _v0 | _d0 | _p0
-------------+-------+-----+-----+------
 Legifrance |  0.95 |  15 |   1 | 0.95
 Wikipedia  |   0.8 |  16 |   1 |  0.8
 Doctissimo |   0.4 |  17 |   1 |  0.4
 Gorafi     |   0.2 |  18 |   1 |  0.2
 Onion      |   0.1 |  19 |   1 |  0.1
(5 rows)
```

Observe that MayBMS has added three additional columns, `_v0`, `_d0`, and `_p0`. Out of these `_v0` and `_p0` are the important ones. `_v0` shows the event ID that defines the tuples, and `_p0` its probability. As we are creating a TID table, there are different independent events for every tuple.

Similarly, for the table Claims:

```
CREATE TABLE ClaimsP AS PICK TUPLES FROM Claims
    INDEPENDENTLY WITH PROBABILITY proba;
SELECT * FROM ClaimsP;
```

```
   source    | claim | proba | _v0 | _d0 | _p0
-------------+-------+-------+-----+-----+-----
 Legifrance |    42 |   0.7 |   8 |   1 | 0.7
 Wikipedia  |    42 |   0.9 |   9 |   1 | 0.9
 Gorafi     |    42 |   0.6 |  10 |   1 | 0.6
 Wikipedia  |    51 |   0.7 |  11 |   1 | 0.7
 Doctissimo |    51 |   0.4 |  12 |   1 | 0.4
 Wikipedia  |    66 |   0.3 |  13 |   1 | 0.3
 Gorafi     |    66 |   0.9 |  14 |   1 | 0.9
(7 rows)
```

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.

Answer.

```
CREATE TABLE T3 AS SELECT T.source FROM TrustP T, ClaimsP C
    WHERE C.source=T.source;
SELECT * FROM T3;
```

```
   source   | _v0 | _d0 | _p0  | _v1 | _d1 | _p1
------------+-----+-----+------+-----+-----+-----
 Doctissimo |  17 |   1 |  0.4 |  12 |   1 | 0.4
 Gorafi     |  18 |   1 |  0.2 |  10 |   1 | 0.6
 Gorafi     |  18 |   1 |  0.2 |  14 |   1 | 0.9
 Legifrance |  15 |   1 | 0.95 |   8 |   1 | 0.7
 Wikipedia  |  16 |   1 |  0.8 |  11 |   1 | 0.7
 Wikipedia  |  16 |   1 |  0.8 |   9 |   1 | 0.9
 Wikipedia  |  16 |   1 |  0.8 |  13 |   1 | 0.3
(7 rows)
```

There are now two sets of `_v`, `_d`, `_p` columns, one set corresponding to the events in the ClaimsP input table, and one set corresponding to the TrustP input table. The annotation in the output tuple represents the conjunction of the corresponding events in the `_v` columns. Notice that the table is no longer a TID table.

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.

Answer. To obtain our final probability values for each source values, we execute

```
SELECT source, conf() FROM T3 GROUP BY source;
```

We obtain the same result as before:

```
   source   |  conf
------------+--------
 Doctissimo |   0.16
 Gorafi     |  0.192
 Legifrance |  0.665
 Wikipedia  | 0.7832
(4 rows)
```

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.

Answer. The query in the relational algebra is:

$$\pi_{\mathbf{claim}}\Big(\sigma_{\mathbf{source}\neq\mathbf{s}}\big((\mathsf{ClaimsP}\bowtie\mathsf{TrustP})\bowtie\rho_{\mathbf{source}\to\mathbf{s}}(\mathsf{ClaimsP}\bowtie\mathsf{TrustP})\big)\Big)$$

It can be written in SQL as:

```
CREATE TABLE T4 AS
SELECT C1.claim FROM ClaimsP C1, ClaimsP C2, TrustP T1, TrustP T2
    WHERE C1.claim = C2.claim AND C1.source != C2.source
    AND C1.source = T1.source AND C2.source = T2.source;
```

The probabilities can be computed as:

```
SELECT claim, conf() FROM T4 GROUP BY claim;
```

This yields the following result:

```
 claim |   conf
-------+----------
    42 | 0.530088
    51 |   0.0896
    66 |   0.0432
(3 rows)
```

The probabilities are easily verified for claims 51 and 66 as the product of the two relevant facts from Claims and the two relevant facts from Trust: all these facts need to be there for the corresponding claim to be part of the query answer. For claim 42, we first compute the probabilities p_1, p_2 and p_3 of the independent conjunctions of each relevant Claim tuple with the corresponding Trust tuple. The final probability is then the probability that two or three of the conjunction events are true, which we evaluate as:

$$p_1p_2(1-p_3) + p_1(1-p_2)p_3 + (1-p_1)p_2p_3 + p_1p_2p_3$$

This yields the probability computed by MayBMS.

4 Hard queries and approximation

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

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

Answer.

```
CREATE TABLE Relevant(claim int, proba float);
INSERT INTO Relevant(claim, proba) VALUES (42, .2), (51, .4), (66, .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 algebra and in SQL, then evaluate it. Is the query true on the input instance?

Answer. In the relational algebra:

$$\pi_{\emptyset}(\mathsf{Trust} \bowtie \mathsf{Claims} \bowtie \mathsf{Relevant})$$

In SQL:

```
SELECT DISTINCT 1 AS b FROM Trust T, Claims C, Relevant R
    WHERE T.source = C.source AND C.claim = R.claim;
```

The query is true.

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.

Answer. We first create the RelevantP probabilistic table:

```
CREATE TABLE RelevantP AS PICK TUPLES FROM Relevant
    INDEPENDENTLY WITH PROBABILITY proba;
```

We then evaluate a query that corresponds to that of the previous question:

```
CREATE TABLE T5 AS
SELECT DISTINCT 1 AS b FROM TrustP T, ClaimsP C, RelevantP R
    WHERE T.source = C.source AND C.claim = R.claim;
```

We then evaluate the query probability:

```
SELECT conf() FROM T5 GROUP BY b;
```

We obtain the result:

```
   conf
----------
 0.563471
(1 row)
```

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.

Answer. The query is true.

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 your machine is doing (press `q` to quit).

> **Answer.** The steps to follow are exactly the same as in Question 3. The computation hangs, it is too long to complete successfully. The reason is that the query corresponds to the hard query H_0 which is intractable to evaluate; unlike the original tables, the new tables are larger, so the computation is too slow. Observe that, by contrast, deterministic query evaluation in the previous question completed instantly.

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.

> **Answer.** Calling Res2 the intermediate probabilistic table created in the previous question to hold the results of the query, the approximate probability can be computed, e.g., as:
>
> ```
> SELECT aconf(.05, .05) FROM Res2 GROUP BY b;
> ```