Querying Bayesian model output with PostgreSQL

Wellington PostgreSQL Users Group

Finlay Thompson

17 September 2015
Bayesian modelling
Bayesian analysis involves converting data and conceptual models into probability distributions.

We interpret probabilities in the broad sense that:

- a probability $p$ is a number between 0 and 1
- where $p = 1$ corresponds to TRUE
- and $p = 0$ corresponds to FALSE
- probabilities measure our confidence in a statement

Note that traditional 20th century statistics understands probabilities as the ratios coming from repeated experiments (think coin tosses).
\[ P(\theta|D) = \frac{P(\theta)P(D|\theta)}{P(D)} \]

\( D \) represents the data, and is typically records in a database.

\( \theta \) represents the parameters of some kind of model.

\( P(D|\theta) \) is the likelihood distribution. Probability of measuring \( D \) given \( \theta \).

\( P(\theta|D) \) is the posterior distribution. It represents the result of fitting the data \( D \) to the model \( \theta \).
We use Monte Carlo Markov chain (MCMC) methods that are slow, and produce lots of output.

MCMC methods are accurate.

The posterior output is in the form of samples from the posterior. In practice this means 4000 values per parameter.

Typically models have hundreds of parameters.
Another advantage of MCMC methods is flexibility.

We use the parameter samples to calculate samples for every desired output.

For example, we can produce 4000 samples for each of approx 1.5 million fishing events.

This output is big and expensive. So we moved it into PostgreSQL.
Storing and querying distributions
The estimates are stored in the form of arrays of integers.

They represent *uncertain* quantities, with each value in the array a realisation from the (unknown) posterior distribution.

The order of the arrays are significant, preserving the correlation structure of the estimates.
CREATE TABLE estimate ( 
  model_id INTEGER REFERENCES model(id),
  effort_id INTEGER NOT NULL,
  observed INTEGER, -- null if effort not observed
  estimate INTEGER[] NOT NULL
);
This results in storing a large quantity of data.

Currently around 12 GB

Need to aggregate the data, so integer array sums!
The standard function for aggregating integer arrays is a bit slow.

It checks the lengths of the arrays, and checks the types of arguments, checks checks checks checks.

I took the library function and ripped the checking out!

Open source for the win!
#include "postgres.h"
#include "fmgr.h"
#include "utils/array.h"

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

Datum int_array_sum(PG_FUNCTION_ARGS);

PG_FUNCTION_INFO_V1(int_array_sum);
Datum int_array_sum(PG_FUNCTION_ARGS) {

  ArrayType * state = PG_GETARG_ARRAYTYPE_P(0);
  ArrayType * new = PG_GETARG_ARRAYTYPE_P(1);

  int numargs = ARR_DIMS(state)[0];
  int * state_ptr = (int *) ARR_DATA_PTR(state);
  int * new_ptr = (int *) ARR_DATA_PTR(new);

  int i;
  for (i = 0; i < numargs; i++)
    state_ptr[i] += new_ptr[i];

  PG_RETURN_ARRAYTYPE_P(state);
}
SET search_path = public;

CREATE OR REPLACE FUNCTION int_array_sum(int[], int[]) RETURNS int[]
AS '$libdir/intarraysum', 'int_array_sum'
LANGUAGE C IMMUTABLE STRICT;

DROP AGGREGATE IF EXISTS sum(int[]);
CREATE AGGREGATE sum (int[]) (  
    SFUNC = int_array_sum,  
    STYPE = int[]  
);
Result is queries that are in the order of 1000 times faster.

The result is much more flexibility in reporting, and happy clients.
Reporting is possible in many different slices:

- Fishing year
- target species of fishers
- reporting areas
- type of vessel

Each of these estimates has an accurate estimate of uncertainty, with published confidence intervals.

For a publicly visible example, see [https://data.dragonfly.co.nz/psc/](https://data.dragonfly.co.nz/psc/)