Approximate answers to aggregation queries

April 11, 2013

We assume we are given a very large database that we are to use for analytics. We are interested in answering aggregation queries, i.e. queries of the form "mean(f(row))". We consider two scenarios: sequential approximation and database summarization. In both scenarios we perform some preprocessing of the database with limited knowledge of the future queries. Our goal in both cases is to answer the queries much faster than would be possible by scanning the whole database.

1 Sequential Approximation

In this approach the preprocessing of the data involves randomly permuting it. If there is a single table, we permute the rows. If there are several tables with a single master key, we permute the key values and permute the tables according to that single permutation. Cases in which there are several tables but they don't share a master key seem to be beyond our grasp at this point.

Once the database has been permuted we can use methods of sequential estimation to generate approximations of the desired value within a specified accuracy.

The size of the sample that is needed to guarantee a given level of accuracy can be calculated using Hoeffding's bounds if the value of $f(x)$ is bounded. However, that bound might be very pessimistic if $f(x)$ has large range but small variance. Sequential estimation can take advantage of such situations. By estimating both the mean and the variance a smaller confidence interval can be secured.

2 Database Summarization

For the purpose of aggregate queries, we can think of a table as a sample from some underlying distribution over row values. Under appropriate conditions we can use an approximation of the distribution as a replacement for the database itself. The goal is to create a summarization that can be used to give approximate answers to all aggregate queries (with some restrictions on $f(x)$).

Lets start by considering a table that consists of a single column X. We denote the fraction of rows in which $X = v$ by $P(X = v)$. We assume that $v \in V$ where V is finite. We measure of information content of X with the *entropy* of the distribution P:

$$H(X) = -\sum_{v \in V} P(X = v) \log_2 P(X = v)$$

$H(x)$ is an upper bound on the average number of bits required to encode the column X. Achieving this requires separating the column into two parts: a a variable length index, each value of which corresponds to one of the values in V, and a table of $|V|$ entries, mapping indexes to values. The idea is that the index for common values will be shorter that the index for rare values. If P varies a lot then the lengths of the indices would be different and the savings will be significant.

Suppose we have two columns X, Y. We define the conditional entropy $H(Y|X) \doteq -\sum_{i,j} P(X=i, Y=j) \log_2 \frac{P(X=i,Y=j)}{P(X=i)}$. Using that definition we can always write

$$H(X,Y) = H(X) + H(Y|X)$$

Consider two extreme cases: in the first case, X and Y are **independent**. In this case $H(Y|X) = H(Y)$ and so

$$H(X,Y) = H(X) + H(Y)$$

At the other extreme, suppose that Y is determined by X. This corresponds to the condition $H(Y|X) = 0$ so that

$$H(X,Y) = H(X)$$

The implication condition is the one familiar to database people. If X implies Y then we can normalize the table by separating Y into a new table and using X as an index into that table.

What does it mean when $H(X,Y) = H(X) + \epsilon$? It means that X *almost* determines Y. More specifically, the average number of bits used to index X is only ϵ smaller than the aveage number of bits needed to index X and Y together. In other words, we need only a small number of bits in order to find Y given X.

3 ϵ-covers