# Mimicking Data By Learning Patterns on Data Constraints

**Kapil Khurana   Vishal Goel**

## Abstract

Testing of database engines on real world databases is a crucial problem for database vendors. However, due to privacy concerns and huge transfer cost, it is not feasible for database clients to send their database to vendors. Thus, vendors have to create their own synthetic database with distribution similar to that of the original(client's side) database. Our project aims to solve the problem of "mimicking the original database" by formulating the scenario as a supervised machine learning problem. The training set is a collection of pairs of queries(run on the original database) and their cardinalities(number of rows returned) and the model learned is a mixture model that estimates cardinality of both observed and unseen queries. We tested our model on two datasets and also compared the performance with neural network. Finally, we suggest a way to generate the synthetic database on which the database engine can be tested.

## 1. Motivation

In the database world, the database vendors need to test their engine on real world databases. However, it is difficult for the vendors to get data from clients due to privacy issues and huge transfer cost. Thus, it becomes imperative for the vendors to generate database on their own that "mimics" the client's database, both qualitatively and quantitatively, provided the client can be expected to part with some kind of "anonymised information" of their database.

One way this can be achieved is by running some queries(called observed queries) on each table at the client side and sending the pairs $(q_i, c_i)$ to the vendor side, where $q_i$ is $i^{th}$ query and $c_i$ is its cardinality (number of rows in the table that satisfy $q_i$). Let $C(q, T)$ denote cardinality when query $q$ is run on table $T$. The aim of the vendor is to create a table $T'$ such that for any observed query $q$, $C(q, T') = C(q, T)$ and for any unseen query $q'$, $C(q', T')$ is as close as possible to $C(q', T)$, as shown in figure 1 as a toy example.

**Our Contribution:** We have divided the problem of mimicking data into two parts: 1) Creating Cardinality Estimation Model(CEM) and 2) Database Generation using CEM. For 1), we used ideas from [4] and ran extensive (due to too many hyper-parameters) experiments comparing our results with their's and neural network. We also identified the problem of good training set data generation and have explained the approaches to it in the Experiments section. For 2), we have given an approach based on mixture model.

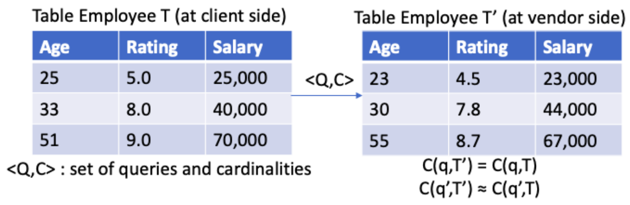

Table Employee T (at client side)

| Age | Rating | Salary |
|---|---|---|
| 25 | 5.0 | 25,000 |
| 33 | 8.0 | 40,000 |
| 51 | 9.0 | 70,000 |

<Q,C> : set of queries and cardinalities

Table Employee T' (at vendor side)

| Age | Rating | Salary |
|---|---|---|
| 23 | 4.5 | 23,000 |
| 30 | 7.8 | 44,000 |
| 55 | 8.7 | 67,000 |

*Figure 1.* Toy Example

## 2. Literature Survey

The most challenging part of the project is cardinality estimation, an open and extremely crucial problem in the database world. The primary aim of a database engine is to efficiently run a query. For this, the query optimiser generates various plans, costs them using the cardinality estimation model and picks the one with the least cost.

The CEMs currently used in the database world are very inefficient and the community has been lately turning to machine learning as an effect. We primarily read papers from four different domains that tackle the problem. Some papers like [1] have used the neural network approach. We reproduced experiments in [1] but the results were unsatisfactory and, moreover, it was hard to use it to generate synthetic database from the model. [2] has used the entropy maximisation approach to find, among all the possible valid tables, the table with the maximum entropy. Though it works well for equality constraints, it was hard to work it out for range predicates. [3] has used the approach of converting the table into a Bayesian Network and then using decision trees on top of it to select values for the attributes, but the approach is hardly scalable. Finally, [4] came up with a mixture model approach, which we have used in our project and explained in Section 4. This approach gives better results than neural network, works for both equality and range predicates and is scalable, as shown in Section 5.

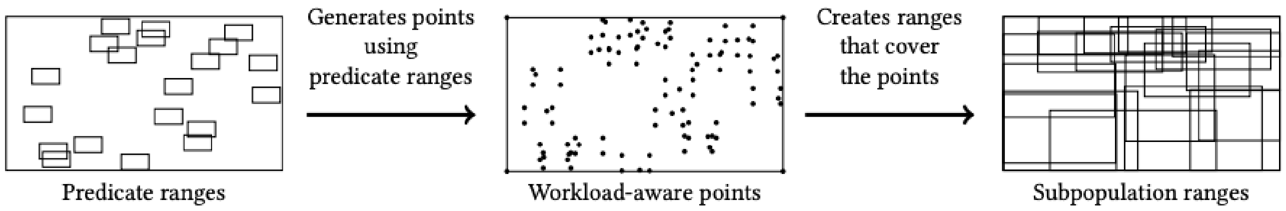

*Figure 2.* Subpopulation Creation

## 3. Problem Statement

**Cardinality Estimation:** Let $T(A_1, .., A_d)$ be a table with $d$ real-valued attributes . Let the tuples in T be denoted by $x_1, ..., x_N$, where each $x_i$ is a d-dimensional vector that belongs to the whole space $B_0 = [l_1, u_1] \times \ldots \times [l_d, u_d]$ where $(l_i, u_i)$ are minimum and maximum values of $i^{th}$ attribute respectively. Let $P_i$ denote the predicate of the $i^{th}$ query. In our experiments, a predicate is a conjunction of one or more range constraints. Thus each query $q$ is of the form *select * from T where* $a_1 \leq A_1 \leq b_1 \wedge ... \wedge a_d \leq A_d \leq b_d$. Note that other kinds of predicates can be easily transformed into the aforementioned form using DeMorgan's Law. Let each predicate $P_i$ be represented by a hyper-rectangle $B_i$. Consider a set of $n$ observed queries $(P_1, c_1), .., (P_n, c_n)$ for $T$ where $f(x)$ denotes pdf of $T$. By definition, we have $\int_{x \in B_i} f(x) = c_i$ for each $i = 1, .., n$. Finally, our goal is to build a model of $f(x)$ that can estimate the cardinality $c'$ of a new predicate $P'$.

**Database Generation:** After getting the CEM, the next step is to generate a synthetic Table $T'$ using CEM. We have given an approach to tackle this problem. However empirical comparison between synthetic and actual database table is part of future work.

## 4. Approach

### 4.1. Mixture Model Approach to Cardinality Estimation

The uniform mixture model [4] represents a population distribution $f(x)$ as a weighted summation of multiple uniform distributions, $g_z(x)$ for $z = 1, ..., m$. Specifically,

$$f(x) = \sum_{z=1}^{m} h(z) g_z(x) = \sum_{z=1} w_z g_z(x)$$

where $h(z)$ is a categorical distribution that determines the weight of the $z$-th subpopulation, and $g_z(x)$ is the pdf (which is a uniform distribution) for the $z$-th subpopulation. The support for $g_z(x)$ is represented by a hyper-rectangle $G_z$. Note that $g_z(x) = 1/|G_z|$ if $x \in G_z$ and 0 otherwise. The selectivity of a predicate $P_i$ is calculated as $\int_{B_i} f(x)dx = \sum_{z=1}^{m} w_z \frac{G_z \cap B_i}{G_z}$. Further, the following operations are performed for creating $G_z$ for $z$ = 1 to $m$.

1. Within each hyper-rectangle, generate $p$ random points.
2. Use random sampling to reduce the number of points to $m$, which serve as the centers of $G_z$ for $z$ = $1, .., m$.
3. Size of each $G_z$ is obtained by taking maximum of distances to the $k$ closest centers.

Note that $p$, $m$ and $k$ are hyper-parameters. In our experiments, $p$ = 10, $m$ = $min$(2000, 2× number of observed queries) and $k$ = 30 gave best results. Figure 2 illustrates how the subpopulations are created.

**Model Training as Optimization:** The optimal parameter **w** is obtained by solving the following optimization problem:

$$\underset{w}{\operatorname{argmin}} \quad \int_{x \in B_0} (f(x) - \frac{1}{|B_0|})^2 dx$$

$$such \quad that \int_{B_i} f(x)dx = c_i, \quad \forall i = 1, .., n$$

$$f(x) \geq 0$$

The approximate solution of the above problem is given by:

$$\mathbf{w}^* = (Q + \lambda A^T A)^{-1} \lambda A c \quad where$$

$$(Q)_{ij} = \frac{|G_i \cap G_j|}{|G_i||G_j|} \quad (A)_{ij} = \frac{|B_i \cap G_j|}{|G_j|}$$

Please refer [4] for the proof of above approximation. Here, $\lambda$ (= $10^5$ in our experiments) is also a hyper-parameter that ensures exact cardinality estimation of observed queries.

### 4.2. Database Generation

After finding hyper-rectangles corresponding to each subpopulation($G_z$) and weight $w$ of each hyper-rectangle, synthetic database can be generated by simply generating $w \times |T|$ points randomly in each hyper-rectangle, where $|T|$ denotes the total number of rows in $T$. This is because the mixture model approach assumes uniform distribution in each hyper-rectangle. Further, it can be easily proved that the cardinality of the original table is equal to the sum of cardinalities in each hyper-rectangle because sum of all $w$'s is equal to 1. Note that more the number of overlaps in a particular region, higher will be the cardinality of that region.

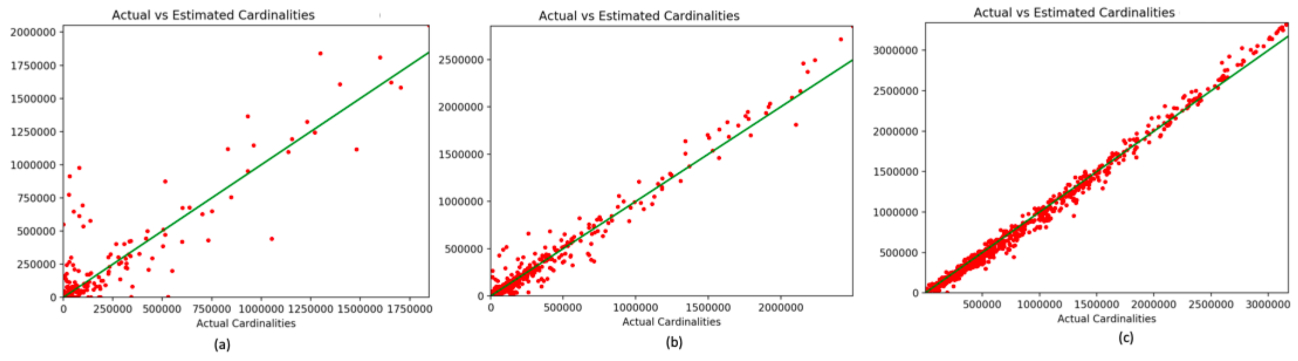

*Figure 3.* Performance on Instacart

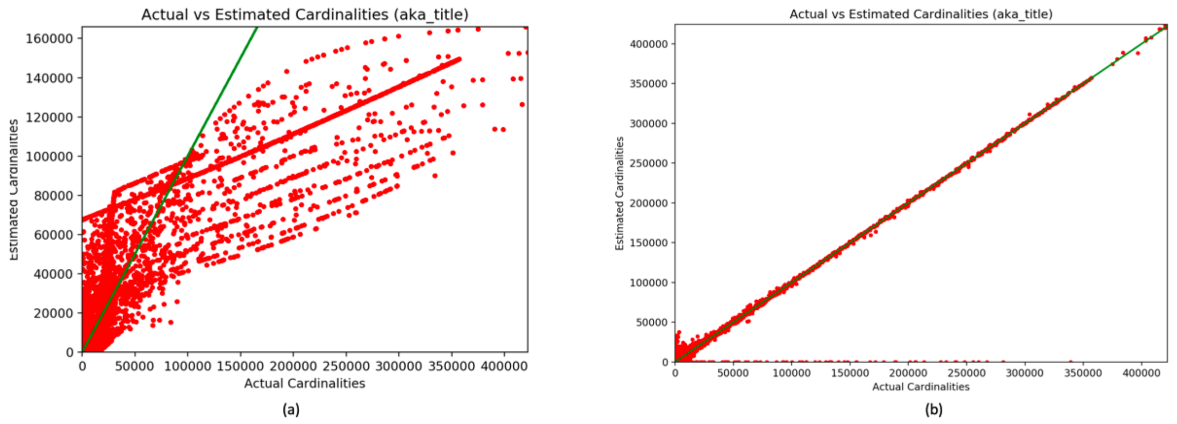

*Figure 4.* Performance on JOB

## 5. Experiments

[4] tests its model on two datasets, DMV and Instacart(explained in [4]). For our experiments, we only chose Instacart out of the two because both datasets are very similar. Instead of DMV, we chose the JOB benchmark (IMDB dataset, explained in [1]). In both, we have used 5-fold cross validation technique.

**Instacart**: Table "Orders" with 3214874 rows has 2 attributes, with ranges (0,23) and (0,31) respectively. 1,500 and 500 queries were used for training and testing respectively. While [4] got a relative error of 7.18%, we scored 34% (figure 3(a)) in our initial implementation. But we observed that apart from the obvious hyper-parameters $(m, p, k)$, creating a good training data set is also crucial, which has not been talked about in [4]. Tuning the so-called "pseudo-hyper-parameters" like size of boxes of the training queries and adding 1-d histogram queries (queries on single attribute) in the training set brought down the error significantly to 24%(figure 3(b)) and then to 10.9%(figure 3(c)) respectively.

**JOB:** The "aka_title" IMDB table contains 425692 rows and has 4 attributes, two of which range in millions. 15,000 and 3,700 queries were used for training and testing respectively. We got a relative error of 21%. We also trained neural network on the same dataset. The comparison has been shown in figure 4. Note that some queries return zero cardinalities because the subpopulations do not cover whole domain space.

## 6. Future Work

The contribution of both the members in the project has been equal. Kapil wrote the code for neural network and Vishal wrote the code for mixture model approach. Both generated queries for the datasets and tuned the models appropriately. As part of the future work, we look forward to tackling the zero-cardinality problem and empirical comparison between synthetic and original database table.

**References**

[1] H. Liu et. al.: Cardinality Estimation Using Neural Networks. In CASCON '15.
[2] Christopher et. al.: Understanding Cardinality Estimation using Entropy Maximization. In PODS 2010.
[3] L. Getoor et. al.: Selectivity Estimation using Probabilistic Models. In SIGMOD 2001.
[4] Y. Park et. al. QuickSel: Quick Selectivity Learning with Mixture Models. In SIGMOD 2018.