Learned Cardinality Estimation: A Design Space Exploration and A Comparative Evaluation

Ji Sun*
Tsinghua University
sun-j16@mails.tsinghua.edu.cn

Jintao Zhang*
Tsinghua University
jtzhang6@gmail.com

Zhaoyan Sun
Tsinghua University
sunzy18@mails.tsinghua.edu.cn

Guoliang Li*
Tsinghua University
liguoliang@tsinghua.edu.cn

Nan Tang
Qatar Computing Research Institute
ntang@hbku.edu.qa

PVLDB, 15(1): 85 - 97, 2022

Cardinality Estimation

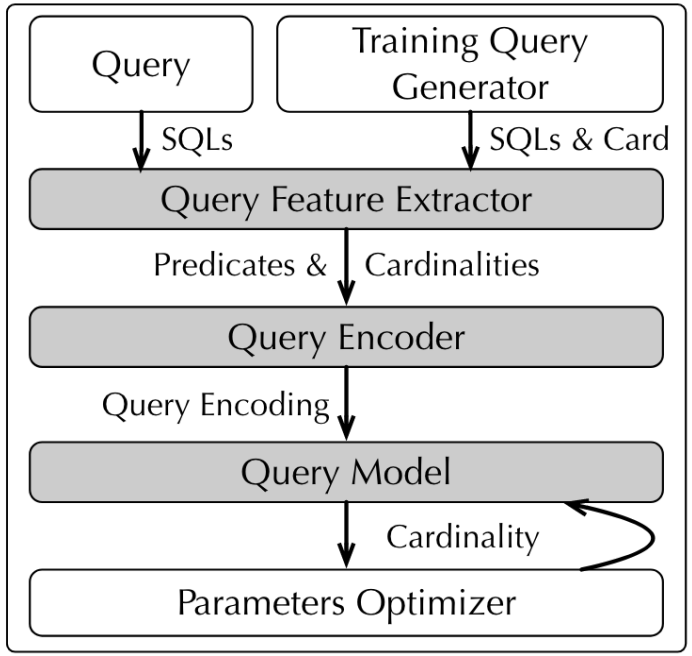
- Non-learned Methods
  - include histograms and samplings
- Learned Query Models
  - learn a mapping function between an SQL query and its cardinality
  - treat cardinality estimation as a typical regression problem
- Learned Data Models
  - learns a joint data distribution of each data point
  - treat cardinality estimation as a density estimation problem
  - divided into unsupervised data models and supervised data models

Goals

- A design space exploration
  - Define a design space of learned solutions for cardinality estimation, categorize these solutions by their shared characteristics, and present a unified workflow to illustrate different design choices.
- A comparative evaluation
  - Conduct a comprehensive evaluation of various learned solutions by varying multiple parameters, summarizing the results to guide practitioners in making informed decisions under different practical scenarios.
- A cardinality-estimation testbed
  - Develop a cardinality-estimation testbed with reusable components that enable researchers and practitioners to design new models for ad-hoc applications with lower design and implementation overhead.

Query Modeling

- **Problem:** learn a mapping model $f(Q)$ between an SQL query Q and its cardinality $|Q(D)|$ on database D.
- **Query Feature Extractor**: decide which features are useful.
- **Query Encoder**: encode all features in a single vector.
- **Query Model**: select appropriate model for modeling the query features, including both traditional statistic-based models and neural networks.



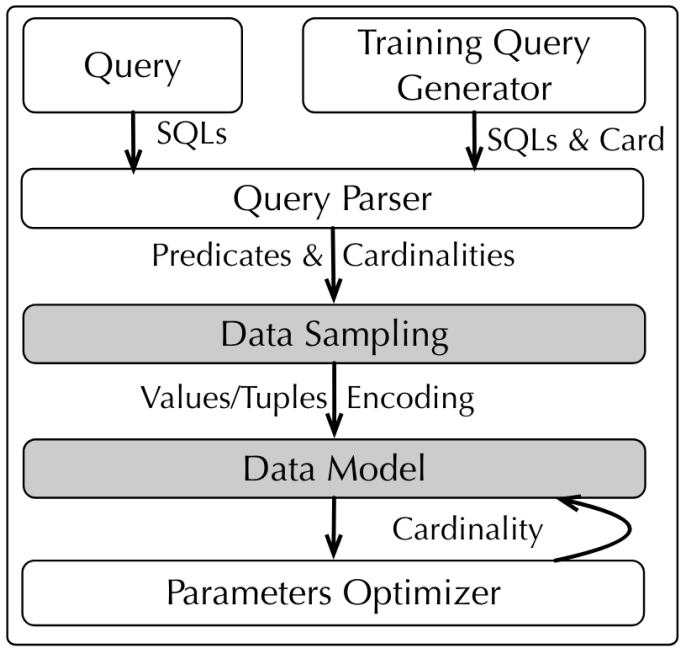
(a) Supervised Query Methods

Data Modeling

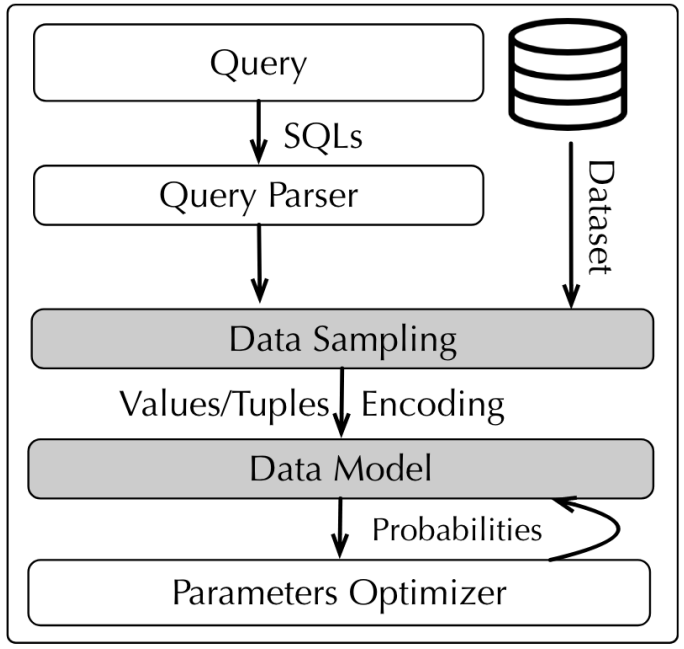
- **Problem**: learn the joint data distribution
- **Supervised data models** learn the data distribution by using some SQL queries and their real cardinalities.
- **Unsupervised data models** directly use the data to train the models. During the training phase, they scan dataset (or data samples) to learn the probability of different values.
- **Inference**. After these models have been trained to learn the data distribution, estimating the cardinality of a query is typically done by uniformly sampling tuples and estimating the cumulative probability of selected tuples from samples.

Data Modeling

- **Query Parser**: apply the query predicates on any data tuple to output 1 if selected; and 0 otherwise.
- **Data Sampling**: sample tuples from dataset.
- **Data Model**: builds a distribution model based on those samples. Probabilistic graphic model, neural network, and statistic-based model.



(b) Supervised Data Methods



(c) Unsupervised Data Methods

SOTA learned cardinality estimators

Data Model

	Model	*Method*	*Parameter Optimizer*	*SQL Parser*	*Sampling*	*Join Decomposition*
Unsupervised	Bayesian Network	Baysian	/	Column Predicates	/	Fanout Scaling
Unsupervised	Autoregressive	NeuroCard	Gradient-based	Column Predicates	Progressive Sampling	Fanout Scaling
Unsupervised	Autoregressive	Naru	Gradient-based	Column Predicates	Progressive Sampling	/
Unsupervised	Autoregressive	DLM	Gradient-based	Column Predicates	Importance Sampling	/
Unsupervised	Sum Product Network	DeepDB	Gradient-based	Column Predicates	/	Fanout Scaling
Supervised	Gaussian Kernel	Feedback-KDE	Gradient-based	Column Predicates	Random Sampling	/
Supervised	Uniform Mixture Model	QuickSel	Analytic-based	Column Ranges	Query-aware Sampling	/

Query Model

	Model	*Method*	*Parameter Optimizer*	*Feature*	*Encoding*
Supervised	XGBoost	LocalXGB	/	Column Ranges	range bounds vector
Supervised	Multi-set Convolutional Network	MSCN	Gradient-based	Join Conditions/Tables/Filter Conditions/samples	one-hot/normalized float/bitmap
Supervised	Recurrent Neural Network	RNN	Gradient-based	Join Conditions/Tables/Filter Conditions	one-hot/normalized float
Supervised	Neural Network	LocalNN	Gradient-based	Column Ranges	range bounds vector

Figure 1: A summary of the SOTA learned cardinality estimators.

Naru

- Autoregressive (AR) model is designed to predict the next value for a given sequence of values. It factorizes the joint distribution as:

$$P(v) = P(v_1)P(v_2|v_1), \cdots, P(v_m|v_1, v_2, \cdots, v_{m-1})$$

- Naru computes cross entropy loss for each output value and input value, and minimizes the mean loss by using gradient-based parameter optimization.

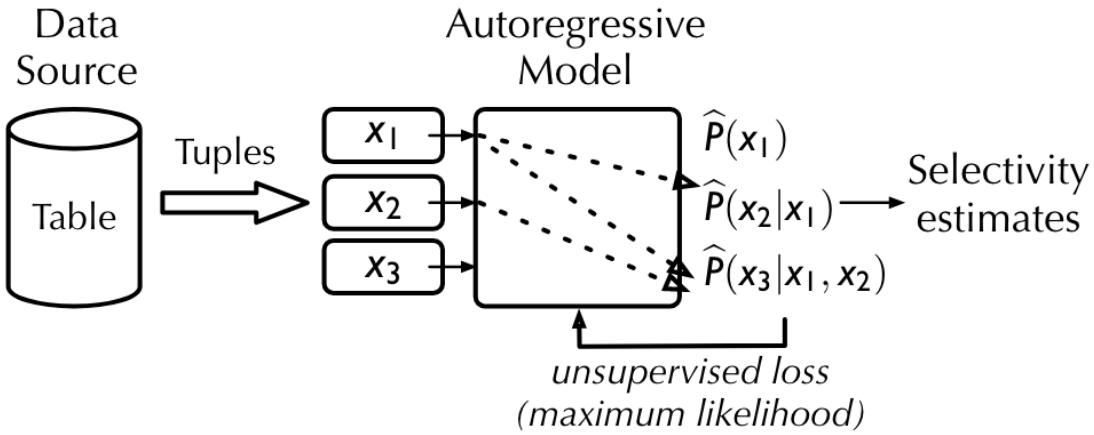


Figure 2: Overview of the estimator framework. Naru is trained by reading data tuples and does not require supervised training queries or query feedback, just like classical synopses.

DeepDB

- Sum Product Network (SPN)

1. DeepDB outer joins all tables as dataset T = T1 ✶ T2 ✶ · · · ✶ Tn .
2. DeepDB recursively splits the dataset (data clustering for rows and correlation identifying for columns).
3. DeepDB learns the weights of edges connecting to sum nodes by fitting the joint probability of data samples, and the weights decide how does each partition contribute to the joint probabilities.

production_year	kind_id	probability
1994	1	0.38
1995	1	0.39
1994	2	0.10
1995	2	0.11

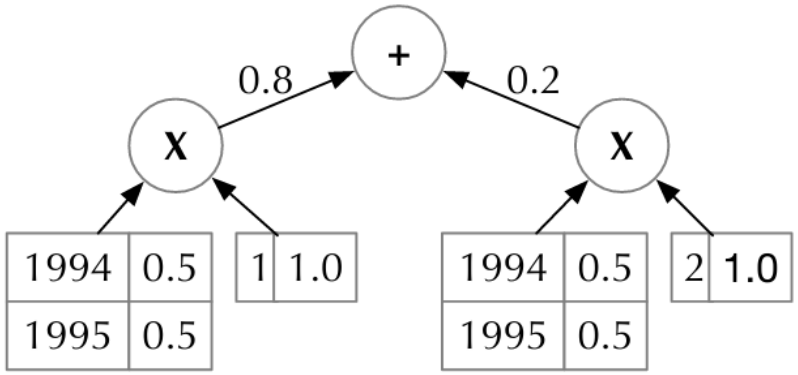


Figure 5: An example of Sum Product Network.

MSCN

- MSCN proposes multi-set convolutional neural network to model SQL queries.
- MSCN divides an SQL query into three sets, including tables in FROM clause, join conditions and filter conditions.

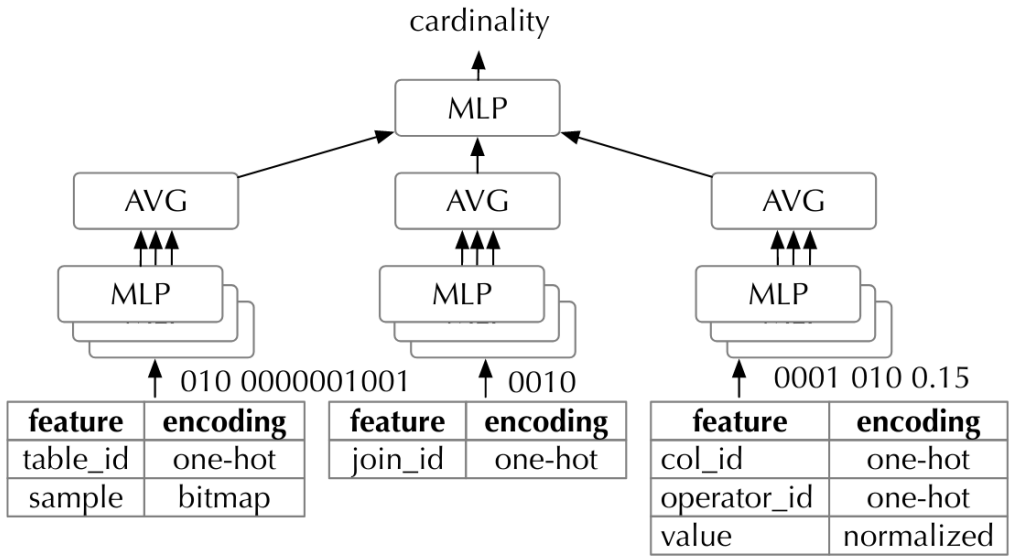


Figure 6: An example of MSCN.

SELECT COUNT(*) FROM title t, movie_companies mc WHERE t.id = mc.movie_id AND t.production_year > 2010 AND mc.company_id = 5

Table set { [0 1 0 1 … 0], [0 0 1 0 … 1] } Join set { [0 0 1 0] } Predicate set { [1 0 0 0 0 1 0 0 0.72], [0 0 0 1 0 0 1 0 0.14] }

table id samples join id column id value operator id

Figure 2: Query featurization as sets of feature vectors.

Datasets

- **Forest and Power**. The data types of them are all integers and can be supported by any method naturally.
- **IMDB**. We select 12 columns from 6 tables. All 6 tables are joined by using key movie_id and id(title).
- **XueTang** is a real-world OLTP benchmark for online education.

Table 2: Statistics of Datasets.

Dataset	#Table	#Rows	#Columns	Domain Size
Forest [11]	1	581K	9	10^{24}
Power [11]	1	2.1M	7	10^{17}
IMDB [20]	6	1.3M-36.2M	12	10^{30}
XueTang [47]	5	8.5M-9.9M	10	10^{31}
Synthetic	256	500K	2-8	10^{2}–10^{32}

Table 3: Synthetic dataset parameters.

#Columns	2, 4, 6, 8, 12	Correlations	0.2, 0.4, 0.6, 0.8
#Distinct Values	10, 100, 1,000, 10,000	Skew	0.2, 0.4, 0.6, 0.8

(2) Correlation *corr* is the probability of each value in one column identical to another. For each pair of column combination, we consider the values in the same position of these two columns to have the same *corr* probability.

(3) Skewness *skew* is a parameter of density distribution $f(x) = (1+x\cdot(skew-1))^{-1-\frac{1}{skew-1}}$. Given a linearly increasing x, the distribution is close to exponential if $skew = 1$; otherwise, the distribution is close to uniform.

Exp-1: Overall Comparisons on Real Datasets

- Unsupervised estimators, Naru and DeepDB, outperform other methods on single real tables Forest and Power. Quicksel fails on both datasets.
- Bayesian produces small median error but large max error, as its conditional independent assumption may fail on some values.
- MSCN outperforms other data model based methods on IMDB that requires to join several tables.

Table 4: Overall Accuracy Comparison on Real Datasets.

Datasets	Forest					Power					IMDB					XueTang				
Methods	median	mean	90%	95%	max	median	mean	90%	95%	max	median	mean	90%	95%	max	median	mean	90%	95%	max
Quicksel	2.73	217	126	731	2e4	6.27	670	598	2e3	4e4	10.4	667	1e3	2e3	3e4	15.6	482	503	1434	3e4
Feedback-KDE	1.11	2.23	3.15	6.12	173	1.10	2.01	4.00	5.34	61	11.5	257	143	366	1e3	32.0	1283	1100	6725	4e4
Bayesian	1.13	2.37	5.60	7.00	1218	1.15	11.2	2.10	3.00	3e4	12.5	306	78.3	521	5e3	1.40	**1.81**	**3.0**	12	230
Naru (NeuroCard)	1.14	**2.24**	3.01	**4.79**	122	1.07	**1.30**	**1.75**	**2.00**	**15.0**	3.85	6.85	9.66	11.2	**477**	**1.26**	2.82	6.0	10.0	30.0
DeepDB	**1.06**	2.51	**2.56**	4.97	117	**1.03**	1.72	2.28	3.76	77.2	4.26	8.28	13.5	25.3	789	1.47	6.23	20.0	30.3	92.7
MSCN	1.91	5.17	12.7	20.0	**96.0**	1.80	5.30	11.2	22.2	84.0	**1.82**	**6.59**	**5.62**	**9.88**	536	1.33	2.33	5.0	**6.0**	**19.0**
LocalNN	1.94	4.64	9.15	13.9	136	1.77	3.88	6.75	11.0	105	9.38	18.3	19.7	22.1	965	4.0	55.0	11.1	17.8	3508
LocalXGB	2.70	7.42	10.9	20.4	511	1.93	3.27	5.29	8.16	73.4	8.12	20.2	18.3	25.3	1e3	3.48	82.3	8.92	17.1	3748

Exp-2: Varying the #Columns

- Naru and DeepDB outperform other methods on accuracy.
- When the number of column increases, because more columns make the data distribution more complicated, and it's harder to fit such data distribution for these models.
- Naru, MSCN and DeepDB still perform well on dataset with 8 columns.

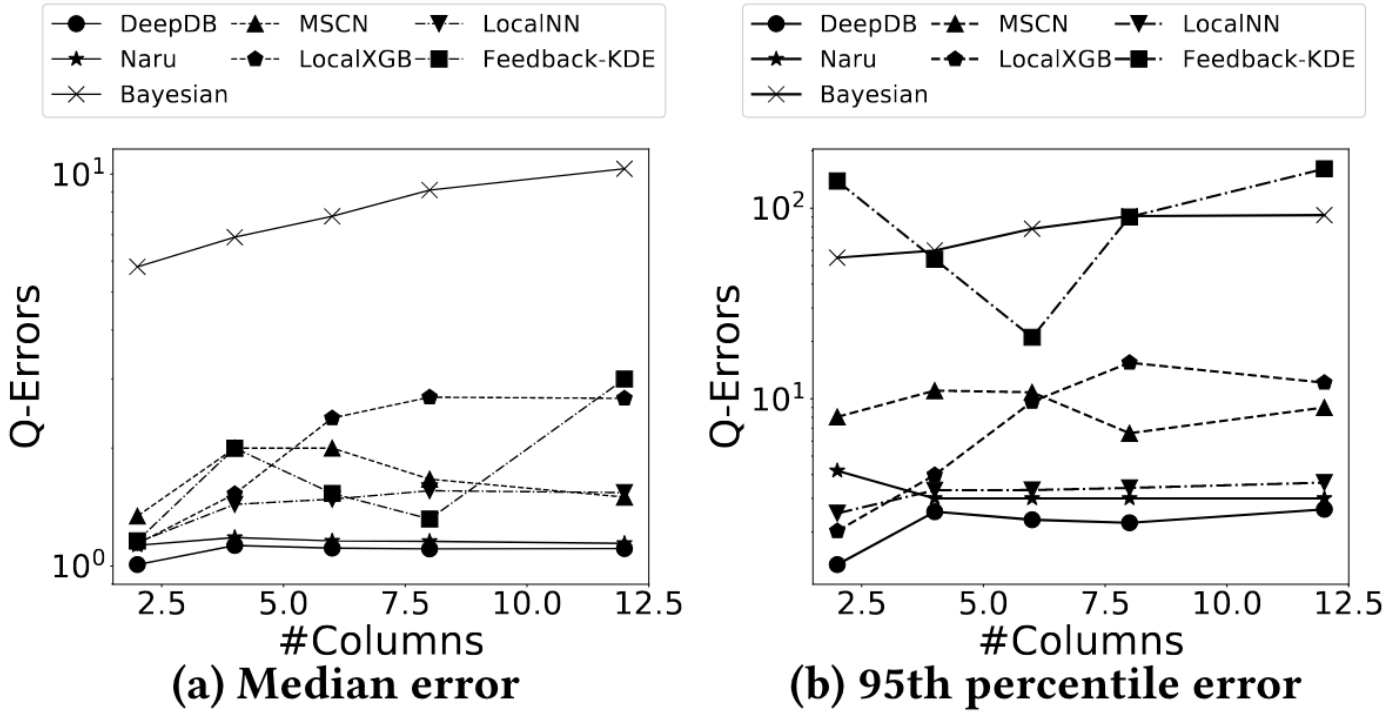


Figure 7: [Synthetic] Cardinality Errors on Varying #Columns (#distinct value=1000, correlation=0.6, skew=0.6).

Exp-3: Varying #Distinct Values

- The accuracy of learned estimators based on query model decreases significantly with domain size increasing.
- DeepDB performs the best among all learned estimators with larger domain size, and the 95th percentile error decreases with domain size increasing.

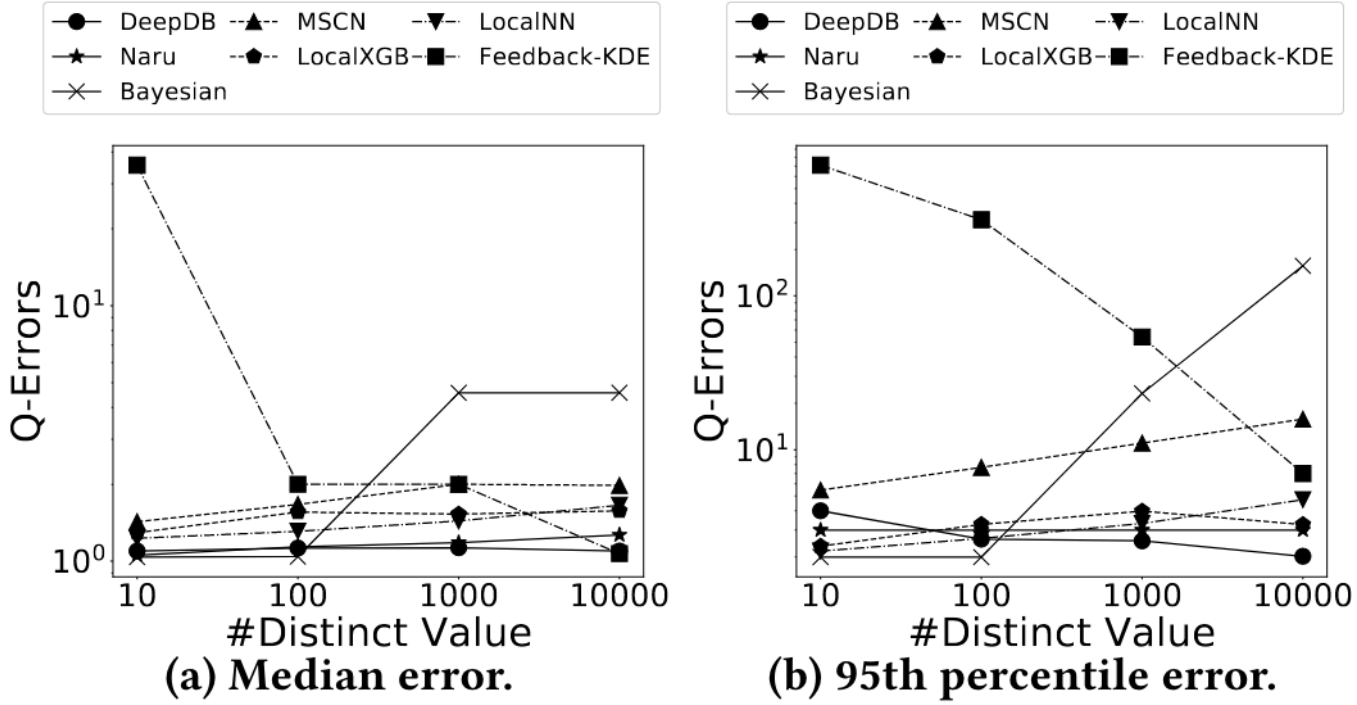


Figure 10: [Synthetic] Cardinality Errors on Varying #Distinct Values (correlation=0.6, column=4, skew=0.6).

Exp-4: Varying Correlations.

- The accuracy of most of the methods decreases when data correlation becomes larger.
- Bayesian can search the optimal probabilistic graph and fit larger correlations.
- Naru outperforms other estimators on dataset with high correlation because autoregressive model uses lossless distribution factorization to fit the dataset.

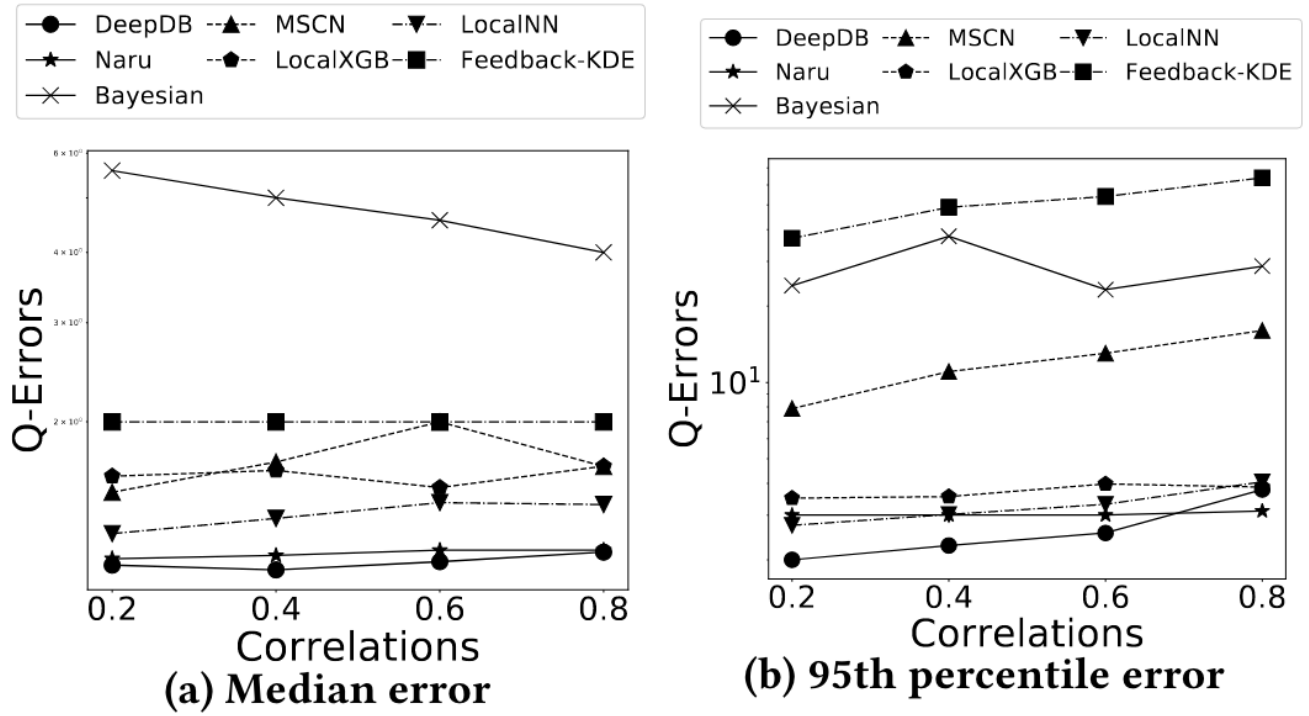


Figure 12: [Synthetic] Cardinality Errors on Varying Correlations (#distinct value=1000, column=4, skew=0.6).

Exp-5: Varying Skewness.

- Median and 95th percentile errors of Naru and DeepDB increase with skewness increasing.

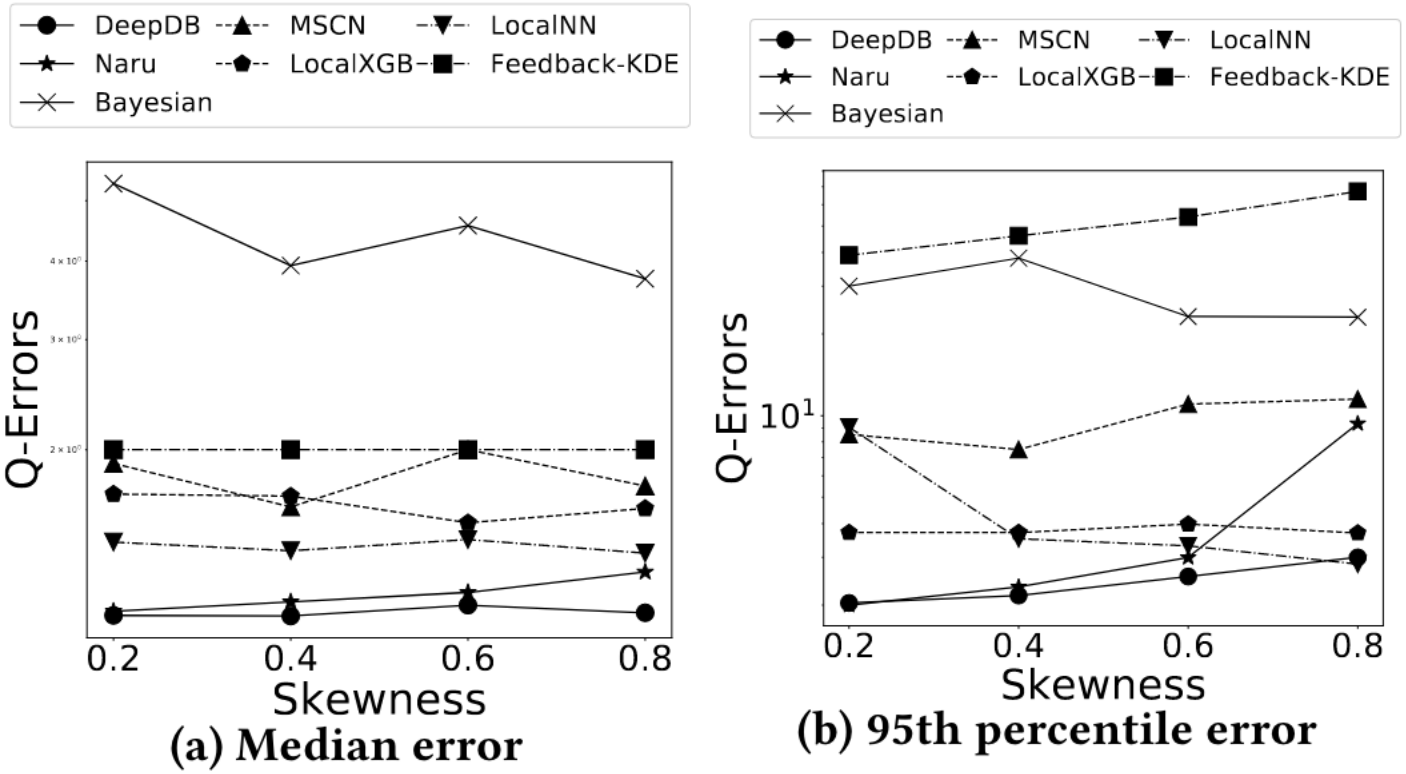
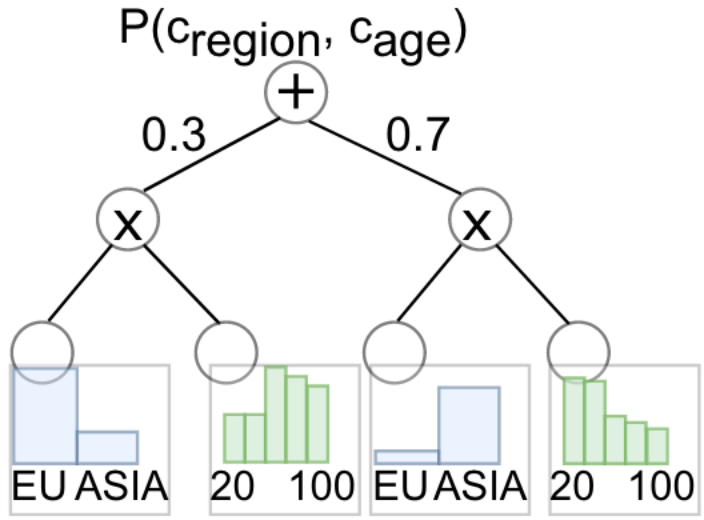


Figure 13: [Synthetic] Cardinality Errors on Varying Skews (#distinct value=1000, column=8, correlation=0.6).



Deepdb

Exp-6: Training Time and Estimation Time

- All the methods spend more training time with the number of columns increasing.

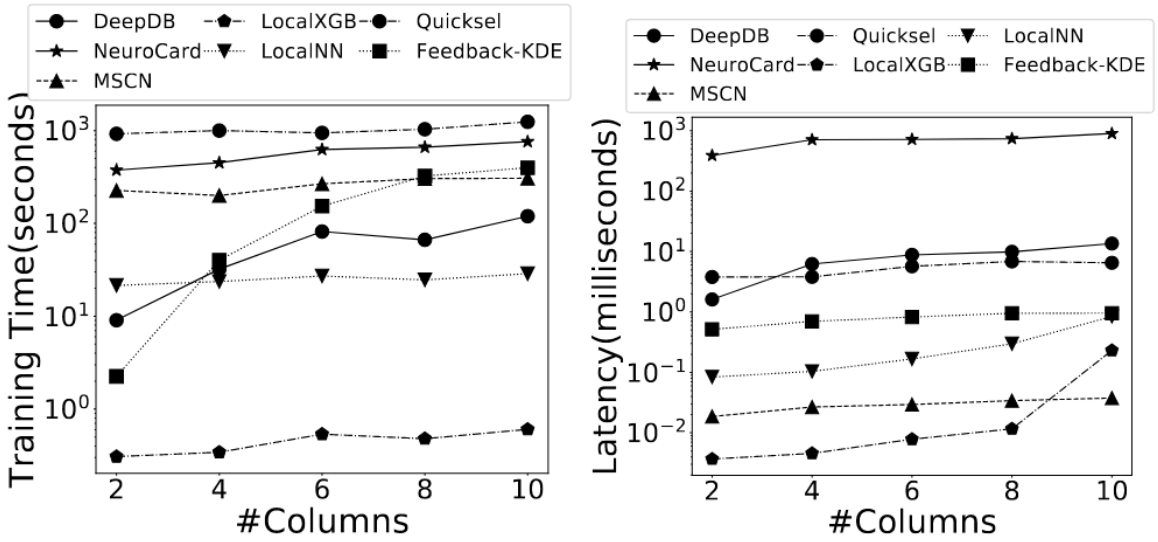
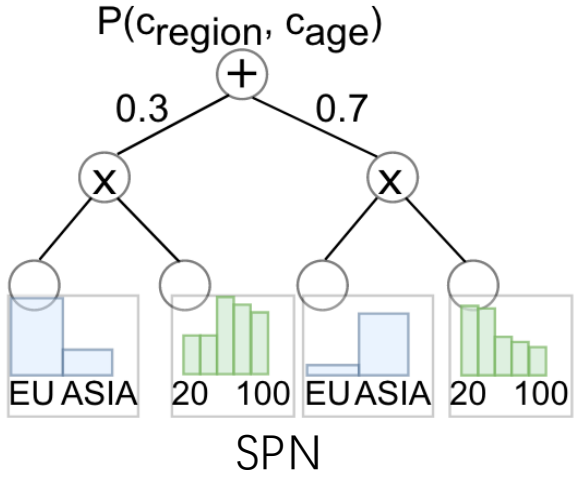
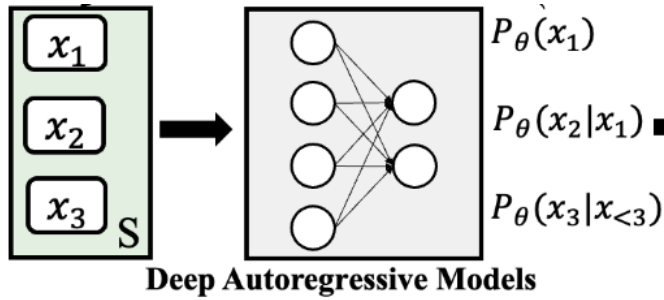


Figure 19: [XueTang] Training Time.

Figure 20: [XueTang] Estimate Latency.





Summary

(1) Data Models DeepDB and Naru are the most effective methods for single tables.

(2) Query Model MSCN is the most effective for multiple tables.

(3) Query Models are more efficient than Data Models.

(4) Data Models are more robust than Query Models.

(5) Training queries are vital to Query Models.

(6) Samples are crucial to Data Models.

(7) Estimators based on neural network are more accurate than statistic-based estimators.

(8) Statistic-based query model is the most efficient.