The Relational Data Borg is Learning

fdbresearch.github.io relational.ai

Dan Olteanu

University of Zurich

VLDB 2020 Keynote
Virtual Tokyo, Sept 1, 2020

Acknowledgments

FDB team, in particular:


Ahmet

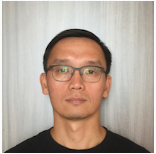

Amir

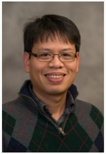

Haozhe


Max


Milos

RelationalAI team, in particular:


Hung


Long


Mahmoud


Molham

Database Research In Data Science Era

Reasons for DB research community to feel empowered:

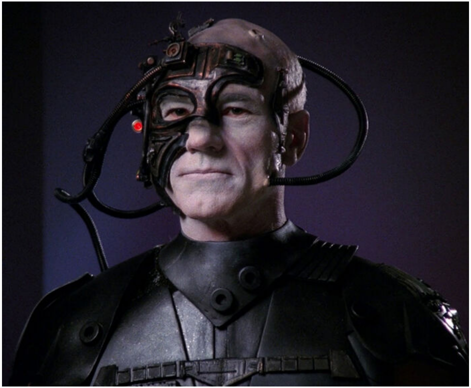
1. Pervasiveness of relational data in data science
   - Hard fact
2. Widespread need for efficient data processing
   - Core to our community's *raison d'être*
3. New processing challenges posed by data science workloads
   - DB's approach reminiscent of Star Trek's Borg Collective

These reasons also serve as motivation for our work.

Star Trek Borg

Co-opt technology and knowledge of alien species

to become ever more powerful and versatile



Relational Data Borg

Assimilate ideas and applications of related fields

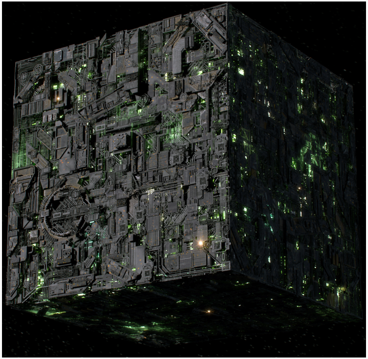
to adapt to new requirements and

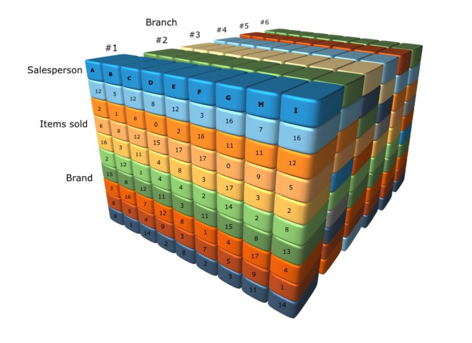
become ever more powerful and versatile

Unlike in Star Trek, the Relational Data Borg

- moves fast
- has great skin complexion and
- is reasonably happy

Borg Cube vs Data Cube

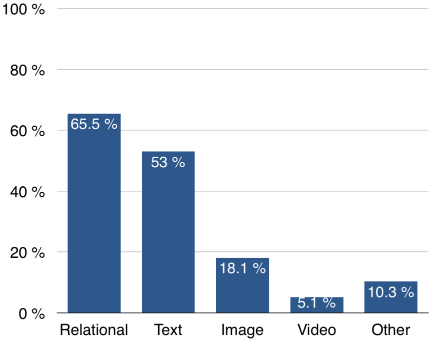




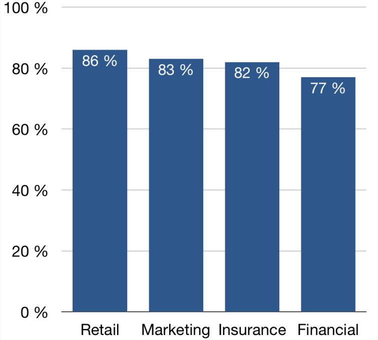
Resistance IS futile in either case

Relational Data is Ubiquitous

Kaggle Survey: Most Data Scientists use Relational Data at Work!



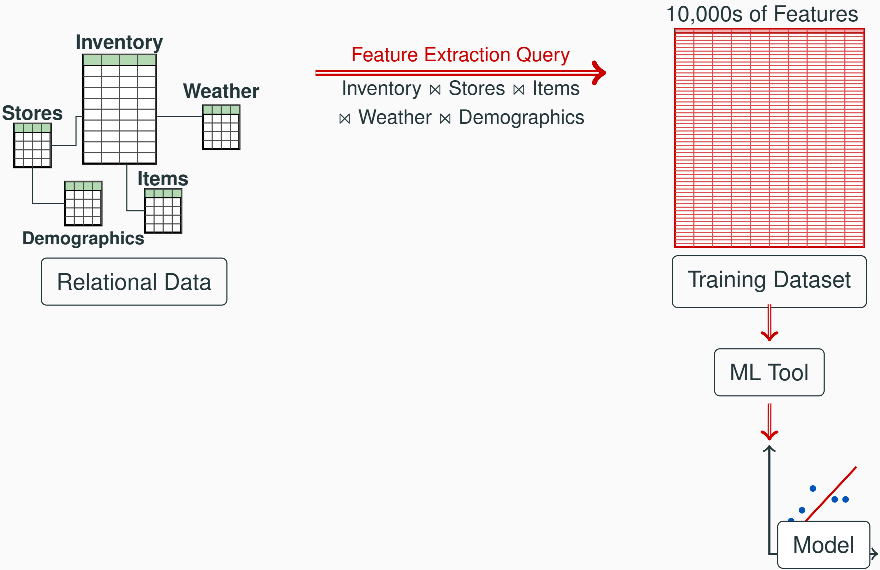
Overall



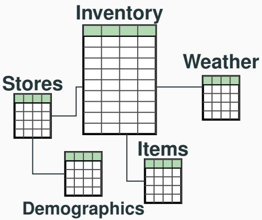
By Industry

Source: The State of Data Science & Machine Learning 2017, Kaggle, October 2017
(based on 2017 Kaggle survey of 16,000 ML practitioners)

State of Affairs in Learning over Relational Data



State of Affairs in Learning over Relational Data

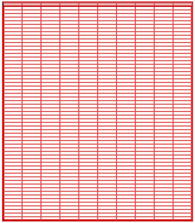


Relational Data

Feature Extraction Query

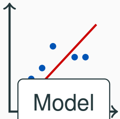
Inventory $\bowtie$ Stores $\bowtie$ Items $\bowtie$ Weather $\bowtie$ Demographics

10,000s of Features



Training Dataset

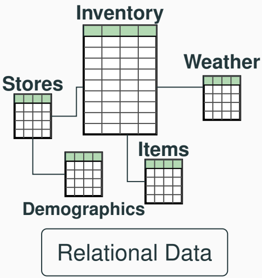
ML Tool



Model

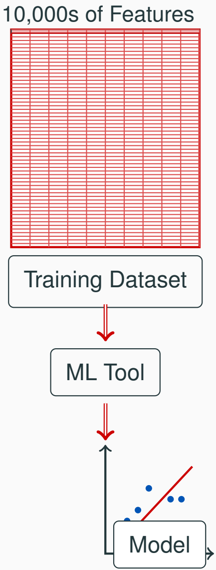
Structure-Agnostic Learning:

State of Affairs in Learning over Relational Data



Feature Extraction Query

Inventory $\bowtie$ Stores $\bowtie$ Items $\bowtie$ Weather $\bowtie$ Demographics

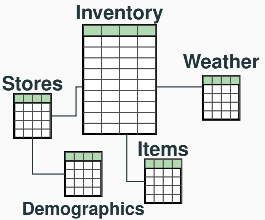


Structure-Agnostic Learning:

1. Unnecessary data matrix materialization

   Relational structure carefully crafted by domain experts thrown away

State of Affairs in Learning over Relational Data

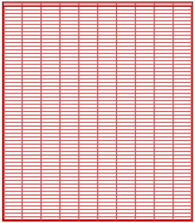


Relational Data

Feature Extraction Query

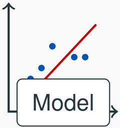
Inventory $\bowtie$ Stores $\bowtie$ Items $\bowtie$ Weather $\bowtie$ Demographics

10,000s of Features



Training Dataset

ML Tool



Model

Structure-Agnostic Learning:

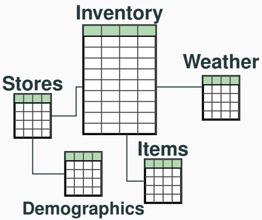
1. Unnecessary data matrix materialization

   Relational structure carefully crafted by domain experts thrown away

2. Expensive data move

   Training dataset can be order-of-magnitude larger than the input DB

State of Affairs in Learning over Relational Data

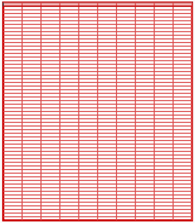


Relational Data

Feature Extraction Query

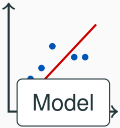
Inventory ⋈ Stores ⋈ Items ⋈ Weather ⋈ Demographics

10,000s of Features



Training Dataset

ML Tool



Structure-Agnostic Learning:

1. Unnecessary data matrix materialization

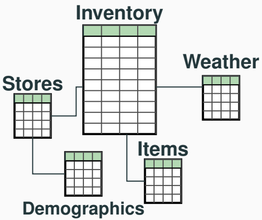
   Relational structure carefully crafted by domain experts thrown away

2. Expensive data move

   Training dataset can be order-of-magnitude larger than the input DB

3. Bloated one-hot encoding

State of Affairs in Learning over Relational Data

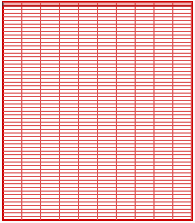


Relational Data

Feature Extraction Query

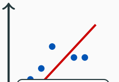
Inventory $\bowtie$ Stores $\bowtie$ Items $\bowtie$ Weather $\bowtie$ Demographics

10,000s of Features



Training Dataset

ML Tool



Model

Structure-Agnostic Learning:

1. Unnecessary data matrix materialization

   Relational structure carefully crafted by domain experts thrown away

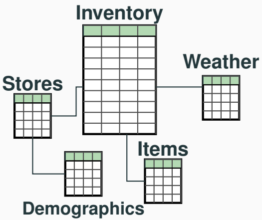
2. Expensive data move

   Training dataset can be order-of-magnitude larger than the input DB

3. Bloated one-hot encoding
4. High maintenance cost

   Recomputation from scratch after updates

State of Affairs in Learning over Relational Data

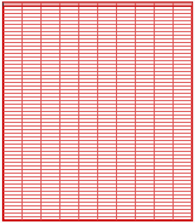


Relational Data

Feature Extraction Query

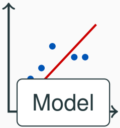
Inventory $\bowtie$ Stores $\bowtie$ Items $\bowtie$ Weather $\bowtie$ Demographics

10,000s of Features



Training Dataset

ML Tool



Structure-Agnostic Learning:

1. Unnecessary data matrix materialization

   Relational structure carefully crafted by domain experts thrown away

2. Expensive data move

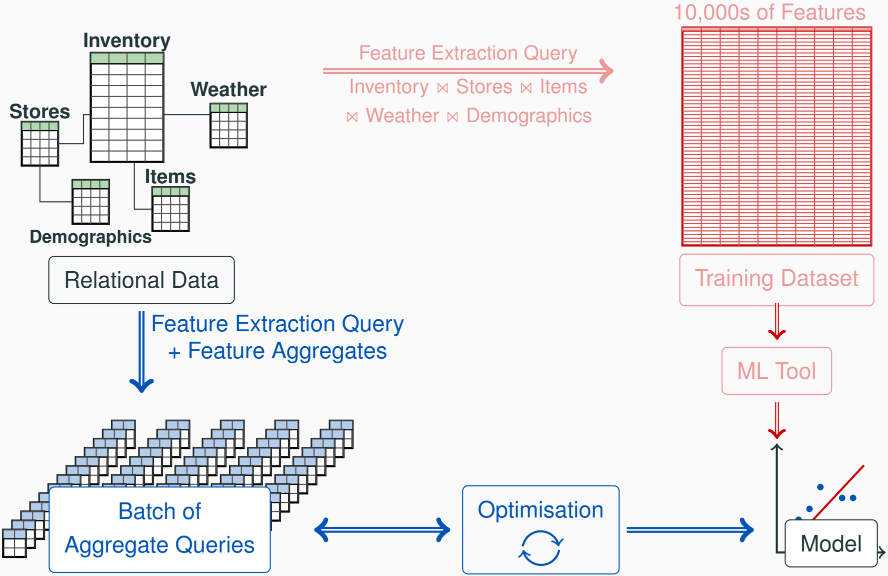
   Training dataset can be order-of-magnitude larger than the input DB

3. Bloated one-hot encoding
4. High maintenance cost

   Recomputation from scratch after updates

5. Limitations inherited from both DB and ML tools

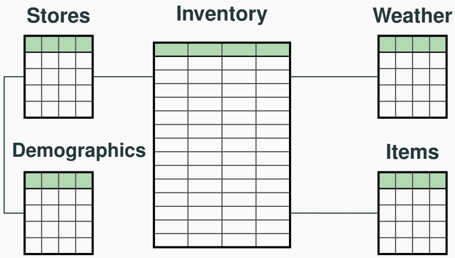
Structure-Aware Learning over Relational Data



Conjecture

The learning time and accuracy of the model can be drastically improved by exploiting the structure and semantics of the underlying multi-relational database.

Structure-aware Learning FASTER than Feature Extraction Query alone



Relation	Cardinality	Arity (Keys+Values)	File Size (CSV)
Inventory	84,055,817	3 + 1	2 GB
Items	5,618	1 + 4	129 KB
Stores	1,317	1 + 14	139 KB
Demographics	1,302	1 + 15	161 KB
Weather	1,159,457	2 + 6	33 MB
Join	84,055,817	3 + 41	23GB

Structure-aware versus Structure-agnostic Learning

Train a linear regression model to predict *inventory* given all features

	PostgreSQL+TensorFlow	
	Time	**Size (CSV)**
Database	–	2.1 GB
Join	152.06 secs	23 GB
Export	351.76 secs	23 GB
Shuffling	5,488.73 secs	23 GB
Query batch	–	–
Grad Descent	7,249.58 secs	–
Total time	13,242.13 secs	

Structure-aware versus Structure-agnostic Learning

Train a linear regression model to predict *inventory* given all features

	PostgreSQL+TensorFlow		Our approach (SIGMOD'19)	
	Time	**Size (CSV)**	**Time**	**Size (CSV)**
Database	–	2.1 GB	–	2.1 GB
Join	152.06 secs	23 GB	–	–
Export	351.76 secs	23 GB	–	–
Shuffling	5,488.73 secs	23 GB	–	–
Query batch	–	–	6.08 secs	37 KB
Grad Descent	7,249.58 secs	–	0.05 secs	–
Total time	13,242.13 secs		6.13 secs	

$2,160\times$ faster while being more accurate (RMSE on 2% test data)

Structure-aware versus Structure-agnostic Learning

Train a linear regression model to predict *inventory* given all features

	PostgreSQL+TensorFlow		Our approach (SIGMOD'19)	
	Time	**Size (CSV)**	**Time**	**Size (CSV)**
Database	–	2.1 GB	–	2.1 GB
Join	152.06 secs	23 GB	–	–
Export	351.76 secs	23 GB	–	–
Shuffling	5,488.73 secs	23 GB	–	–
Query batch	–	–	6.08 secs	37 KB
Grad Descent	7,249.58 secs	–	0.05 secs	–
Total time	13,242.13 secs		6.13 secs	

$2,160\times$ faster while being more accurate (RMSE on 2% test data)

TensorFlow trains one model. Our approach takes < 0.1 sec for any extra model over a subset of the given feature set.

TensorFlow's Behaviour is the Rule, not the Exception!

Similar behaviour (or outright failure) for more:

- **datasets**: Favorita, TPC-DS, Yelp, Housing
- **systems**:
  - used in industry: R, scikit-learn, Python StatsModels, mlpack, XGBoost, MADlib
  - academic prototypes: Morpheus, libFM
- **models**: decision trees, factorisation machines, k-means, ..

This is to be contrasted with the scalability of DBMSs!

How to achieve this performance improvement?

Idea 1: Turn the ML Problem into a DB Problem

Through DB Glasses, Everything is a Batch of Queries

Workload	Query Batch
Linear Regression	SUM(X_i*X_j)
Covariance Matrix	SUM(X_i) GROUP BY X_j
	SUM(1) GROUP BY X_i, X_j
Decision Tree Node	VARIANCE(Y) WHERE $X_j = c_j$
Mutual Information	SUM(1) GROUP BY X_i
Rk-means	SUM(1) GROUP BY X_j
	SUM(1) GROUP BY $\text{Center}_1, \ldots, \text{Center}_k$

Through DB Glasses, Everything is a Batch of Queries

Workload	Query Batch	
Linear Regression	SUM($X_i * X_j$)	[WHERE $\sum_k X_k * w_k < c$]
Covariance Matrix	SUM(X_i) GROUP BY X_j	[WHERE ...]
(Non)poly. loss	SUM(1) GROUP BY X_i, X_j	[WHERE ...]
Decision Tree Node	VARIANCE(Y) WHERE $X_j = c_j$	
Mutual Information	SUM(1) GROUP BY X_i	
Rk-means	SUM(1) GROUP BY X_j	
	SUM(1) GROUP BY $\text{Center}_1, \ldots, \text{Center}_k$	

Through DB Glasses, Everything is a Batch of Queries

Workload	Query Batch		# Queries
Linear Regression	$\texttt{SUM}(X_i * X_j)$	$[\,\texttt{WHERE}\ \sum_k X_k * w_k < c\,]$	814
Covariance Matrix	$\texttt{SUM}(X_i)\ \texttt{GROUP BY}\ X_j$	$[\,\texttt{WHERE}\ \ldots]$	
(Non)poly. loss	$\texttt{SUM(1)}\ \texttt{GROUP BY}\ X_i, X_j$	$[\,\texttt{WHERE}\ \ldots]$	
Decision Tree Node	$\texttt{VARIANCE}(Y)\ \texttt{WHERE}\ X_j = c_j$		3,141
Mutual Information	$\texttt{SUM(1)}\ \texttt{GROUP BY}\ X_i$		56
Rk-means	$\texttt{SUM(1)}\ \texttt{GROUP BY}\ X_j$		41
	$\texttt{SUM(1)}\ \texttt{GROUP BY}\ \text{Center}_1, \ldots, \text{Center}_k$		

(# Queries shown for Retailer dataset with 39 attributes)

Queries in a batch:

- Same aggregates but over different attributes
- Expressed over the same join of the database relations

AMPLE opportunities for sharing computation in a batch.

Models under Consideration

So far:

- Polynomial regression
- Factorisation machines
- Classification/regression trees
- Mutual information
- Chow Liu trees
- *k*-means clustering
- *k*-nearest neighbours
- (robust, ordinal) PCA
- SVM

On-going:

- Boosting regression trees
- AdaBoost
- Sum-product networks
- Random forests
- Logistic regression
- Linear algebra:
  - QR decomposition
  - SVD
  - low-rank matrix factorisation

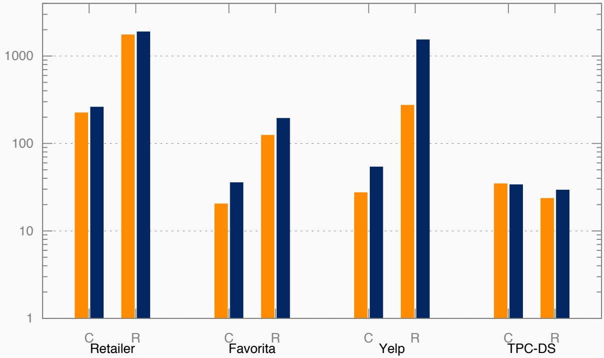
All these cases can benefit from **structure-aware computation**

Natural Attempt:

Use Existing DB System to Compute Query Batch

Existing DBMSs are NOT Designed for Query Batches

Relative Speedup for **Our Approach** over **DBX** and **MonetDB**

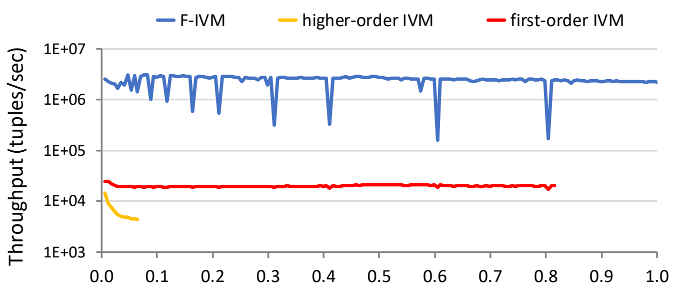


C = Covariance Matrix; R = Regression Tree Node; AWS d2.xlarge (4 vCPUs, 32GB)

Existing DSMSs are NOT Designed for Query Batches

Task: Maintain the covariance matrix over Retailer

- Round-robin insertions in all relations
- All maintenance strategies implemented in DBToaster



Azure DS14, Intel Xeon, 2.40GHz, 112GB, 1 thread; one hour timeout