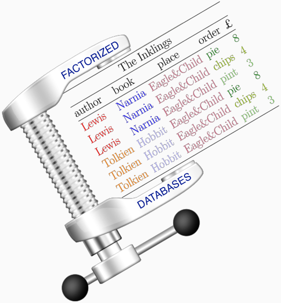
Learning Models over Relational Databases

fdbresearch.github.io relational.ai

Dan Olteanu

Oxford & relationalAI

FG DB Symposium 2020
Darmstadt, March 2020



Acknowledgments

FDB team, in particular:

Jakub

Max

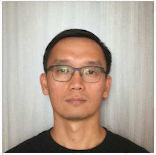
Milos

Ahmet

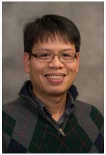
Amir

relationalAI team, in particular:

Mahmoud



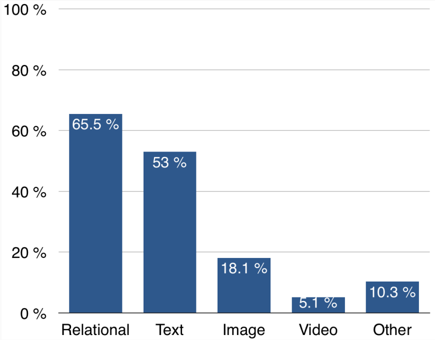
Hung



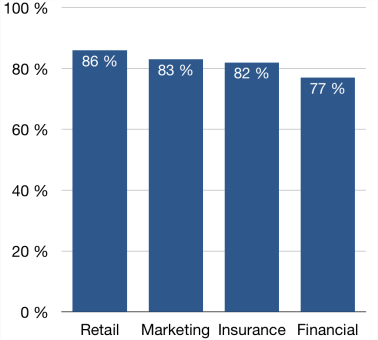
Long

Motivation: Relational Data is Ubiquitous

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



Overall

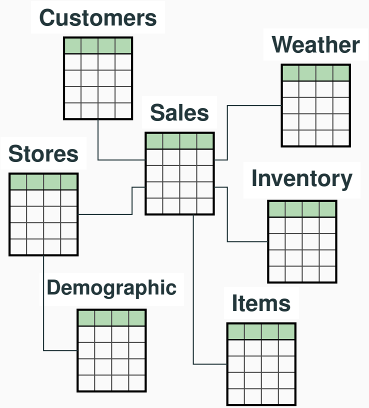


By Industry

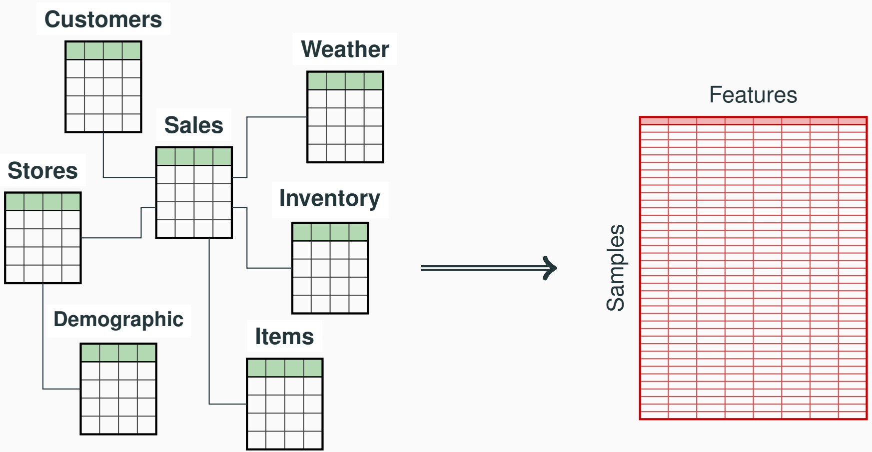
Source: The State of Data Science & Machine Learning 2017, Kaggle, October 2017
(based on 2017 Kaggle survey of 16,000 ML practitioners)

Relational Model: Jewel in the Data Management Crown

- Massive adoption of the Relational Model in last decades
- Many human hours invested in building relational models
- Relational databases are rich with knowledge of the underlying domains



Current State of Affairs in Analytics Workloads



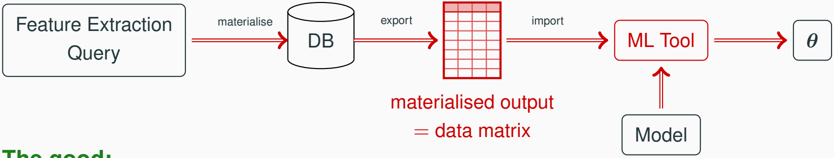
- Carefully crafted by domain experts
- Comes with relational structure
- Throws away relational structure
- Can be order-of-magnitude larger

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.

Current Landscape for ML over DB

No integration



The good:

1. Most DB+ML solutions operate in this space
2. Supports virtually any ML task
3. ML & DB distinct tools on the technology stack

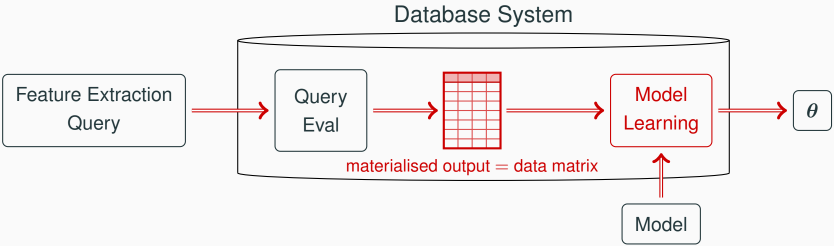
The bad:

1. Materialisation of feature extraction query
2. DB exports data as one table, ML imports it in own format
3. One/multi-hot encoding of categorical variables

Examples:

PostgreSQL + R, Pandas + scikit-learn/TensorFlow, SparkSQL + MLlib, etc.

Loose integration



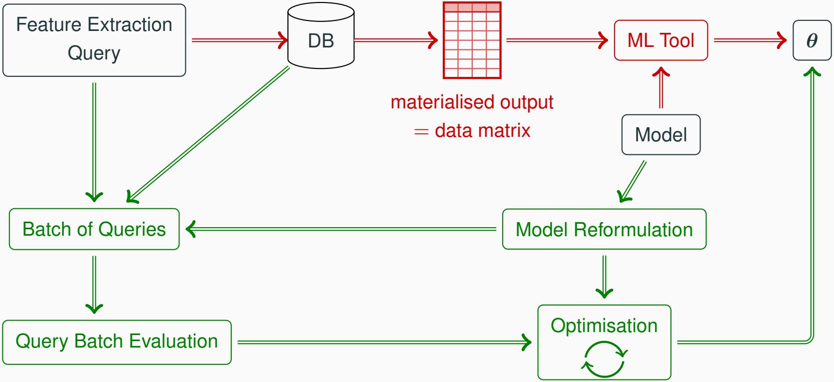
- DB supports ML tasks as UDF
- Same running process for DB and ML
- DB computes one table, ML works directly on it $\rightarrow$ No data export/import

Examples:

MadLib supports comprehensive library of ML UDFs

Bismark gives unified programming architecture for incremental gradient descent

Tight integration

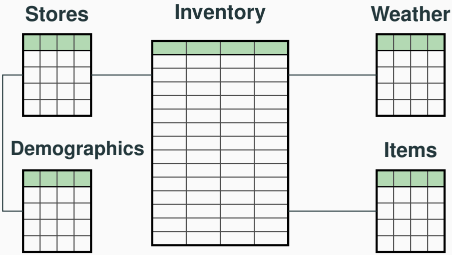


Structure-Aware Learning vs. **Structure-Agnostic Learning**

- Exploit relational structure and semantics
- Exploit database optimisations, e.g., push parts of ML tasks past joins
- One evaluation plan for mixed DB and ML workload

Structure-aware Learning FASTER even than Feature Extraction Query!

Case in Point (1): A Retailer Use Case



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	# Queries
Linear Regression Covariance Matrix	SUM(X_i*X_j) SUM(X_i) GROUP BY X_j SUM(1) GROUP BY X_i, X_j	814
Decision Tree (Regression, 1 Node)	VARIANCE(Y) WHERE $X_j = c_j$	3,141
R*k*-means	SUM(1) GROUP BY X_j SUM(1) GROUP BY $\text{Center}_1, \ldots, \text{Center}_k$	41

(# 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**

Case in Point (2): Ridge Linear Regression

Ridge Linear Regression

$\Downarrow$

Query Batch

Recap: Ridge Linear Regression

Linear regression model:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \langle \boldsymbol{\theta}, \mathbf{x} \rangle = \theta_0 x_0 + \theta_1 x_1 + \dots$$

- Training dataset D defined by *feature extraction query*
  - A tuple $(\mathbf{x}, y) \in D$ consists of feature vector $\mathbf{x}$ and response y
- Parameters $\boldsymbol{\theta}$ obtained by minimising the objective function:

$$J(\boldsymbol{\theta}) = \overbrace{\frac{1}{2|D|} \sum_{(\mathbf{x},y)\in D} \left(\langle \boldsymbol{\theta}, \mathbf{x} \rangle - y\right)^2}^{\text{least square loss}} + \overbrace{\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2}^{\ell_2-\text{regulariser}}$$

From Optimisation to Query Batch

We can solve $\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ with batch-gradient descent:

```
repeat until convergence:
```

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \cdot \nabla J(\boldsymbol{\theta})$$

Model reformulation idea: Decouple

- data-dependent ($\mathbf{x}$, y) computation from
- data-independent ($\boldsymbol{\theta}$) computation

in the formulations of the objective $J(\boldsymbol{\theta})$ and its gradient $\nabla J(\boldsymbol{\theta})$.

From Optimisation to Query Batch

$$J(\boldsymbol{\theta}) = \frac{1}{2|D|} \sum_{(\mathbf{x},y)\in D} (\langle \boldsymbol{\theta}, \mathbf{x} \rangle - y)^2 + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$$

$$= \frac{1}{2|D|} \left(\boldsymbol{\theta}^\top \underbrace{\left(\sum_{(\mathbf{x},y)\in D} \mathbf{x}\mathbf{x}^\top \right)}_{\boldsymbol{\Sigma}} \boldsymbol{\theta} - 2 \left\langle \boldsymbol{\theta}, \underbrace{\sum_{(\mathbf{x},y)\in D} y \cdot \mathbf{x}}_{\mathbf{c}} \right\rangle + \underbrace{\left(\sum_{(\mathbf{x},y)\in D} y^2 \right)}_{s_Y} \right) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$$

$$= \frac{1}{2|D|} \left(\boldsymbol{\theta}^\top \boldsymbol{\Sigma} \boldsymbol{\theta} - 2 \langle \boldsymbol{\theta}, \mathbf{c} \rangle + s_Y \right) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$$

$$\boldsymbol{\nabla} J(\boldsymbol{\theta}) = \frac{1}{|D|} \left(\boldsymbol{\Sigma} \boldsymbol{\theta} - \mathbf{c} \right) + \lambda \boldsymbol{\theta}$$

Σ, c, s_Y can be Expressed as Batch of Queries

Compute one query for each entry $\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top$ in $\boldsymbol{\Sigma}$:

Σ, c, s_Y can be Expressed as Batch of Queries

Compute one query for each entry $\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top$ in $\mathbf{\Sigma}$:

- x_i, x_j continuous

  SELECT SUM (x_i * x_j) FROM D;

where D is the feature extraction query over the input DB.

Σ, c, s_Y can be Expressed as Batch of Queries

Compute one query for each entry $\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top$ in $\mathbf{\Sigma}$:

- x_i, x_j continuous

  SELECT SUM (x_i * x_j) FROM D;

- $\boldsymbol{x}_i$ categorical, x_j continuous

  SELECT x_i, SUM(x_j) FROM D GROUP BY x_i;

where D is the feature extraction query over the input DB.

Σ, c, s_Y can be Expressed as Batch of Queries

Compute one query for each entry $\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top$ in $\mathbf{\Sigma}$:

- x_i, x_j continuous

  SELECT SUM (x_i * x_j) FROM D;

- $\boldsymbol{x}_i$ categorical, x_j continuous

  SELECT x_i, SUM(x_j) FROM D GROUP BY x_i;

- $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ categorical

  SELECT x_i, x_j, SUM(1) FROM D GROUP BY x_i, x_j;

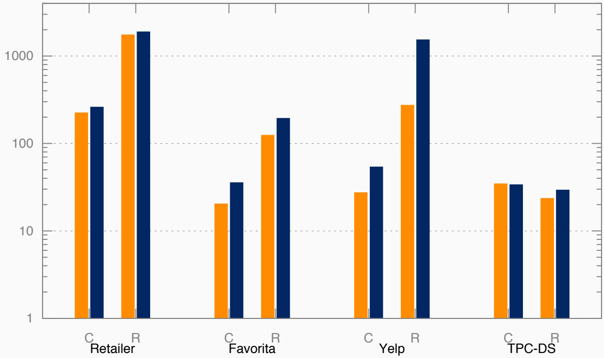
where D is the feature extraction query over the input DB.

Natural Attempt:

Use Existing DB System to Compute Query Batch

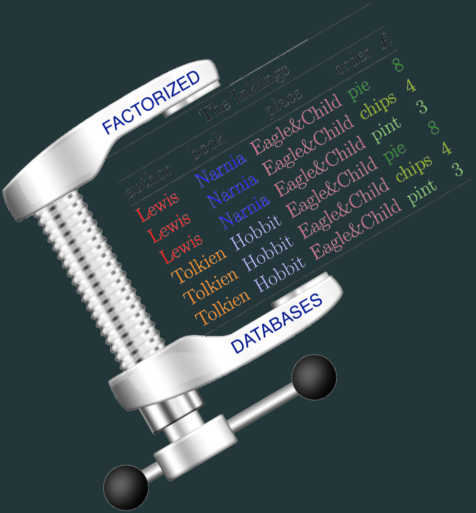
Existing DBMSs are NOT Designed for Query Batches

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



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

Idea 2: Exploit Problem Structure to Lower Complexity



Structure-aware Tools of a Database Researcher

Algebraic structure: (semi)rings $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$

- Distributivity law $\rightarrow$ Factorisation

  Factorised Databases [VLDB'12+'13,TODS'15,SIGREC'16]

  Factorised Machine Learning [SIGMOD'16+'19,DEEM'18,PODS'18+'19, TODS'20]

- Additive inverse $\rightarrow$ Uniform treatment of updates

  Factorised Incremental Maintenance [SIGMOD'18+'20]

- Sum-Product abstraction $\rightarrow$ Same processing for distinct tasks

  DB queries, Covariance matrix, PGM inference, Matrix chain multiplication

  [SIGMOD'18+'19]

Structure-aware Tools of a Database Researcher

Combinatorial structure: query width and data degree measures

- Width measure w for FEQ $\rightarrow$ Low complexity $\tilde{O}(N^w)$

  factorisation width $\geq$ fractional hypertree width $\geq$ sharp-submodular width
  worst-case optimal size and time for factorised joins

  [ICDT'12+'18,TODS'15,PODS'19,TODS'20]

- Degree $\rightarrow$ Adaptive processing depending on high/low degrees

  worst-case optimal incremental maintenance [ICDT'19a, PODS'20]

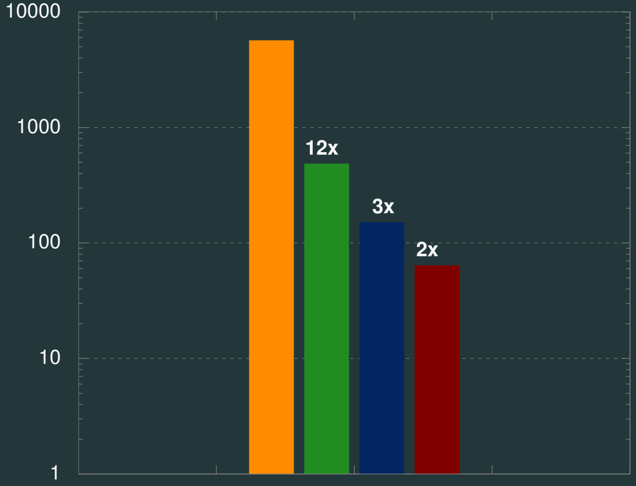
  evaluation of queries with negated relations of bounded degree [ICDT'19b]

- Functional dependencies $\rightarrow$ Learn simpler, equivalent models

  reparameterisation of polynomial regression models and factorisation machines

  [PODS'18,TODS'20]

Idea 3: Lower the Constant Factors



Engineering Tools of a Database Researcher

1. Specialisation for workload and data

   Generate code specific to the query batch and dataset

   Improve cache locality for hot data path

2. Sharing low-level data access

   Aggregates decomposed into views over join tree
   Share data access across views with different output schemas

3. Parallelisation: multi-core (SIMD & distribution to come)

   Task and domain parallelism

[DEEM'18,SIGMOD'19, CGO'20]

Case in Point (3)

Code Optimisations

$\Downarrow$

Non-trivial Speedup

One DSL to Express both DB and ML Workloads! [CGO'20]

Collections are Dictionaries or Sets

- Database relations are modeled as dictionaries

Relation R(A,B)

A	B
a_1	b_1
a_1	b_1
a_2	b_1
a_2	b_1
a_2	b_2

Relation R(A,B) in IFAQ

A	B	$\rightarrow$	$R(A, B)$
a_1	b_1	$\rightarrow$	2
a_2	b_1	$\rightarrow$	2
a_2	b_2	$\rightarrow$	1

Inspired by the FAQ framework [PODS'16]

IFAQ: Iterative Functional Aggregate Queries

- Σ for stateful computation over collection elements:

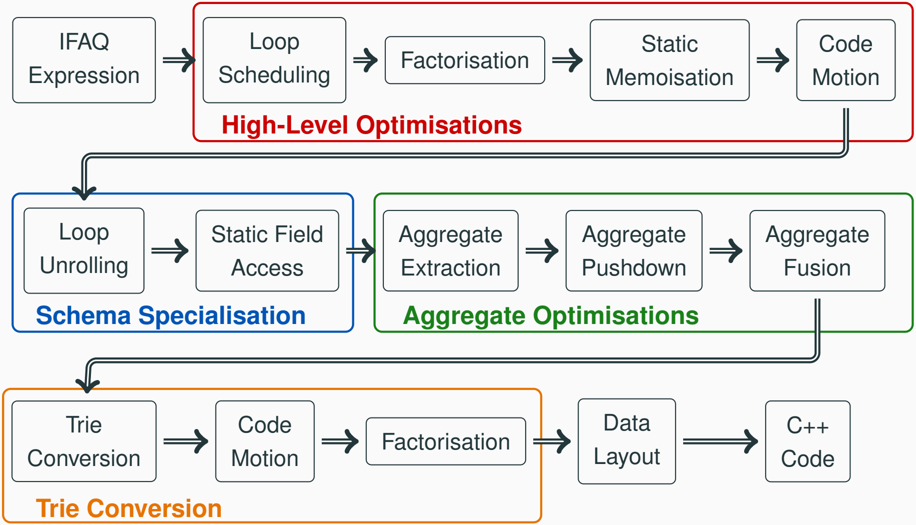
IFAQ		**C++**
$\sum_{e \in set} f(e)$	$\xrightarrow{\textit{Compile}}$	`for(auto& e : set)` `res += f(e);`

- λ for constructing dictionaries:

IFAQ		**C++**
$\lambda_{e \in set} f(e)$	$\xrightarrow{\textit{Compile}}$	`for(auto& e : set)` `res[e] = f(e);`

- Supports while loops and conditionals

Transformation Steps for IFAQ Expressions



Running Example

Dataset with three relations:

Sales(item,store,unit sales) **I**tem(item, price) Sto**R**e(store, city)

Learning Task:

Learn Linear Regression model to predict number of **u**nit sales.

Training Dataset:

$$Q(x) = S(x_S) \bowtie R(x_R) \bowtie I(x_I)$$

(Simplified) Linear Regression in IFAQ

Batch Gradient Descent:

Update $\boldsymbol{\theta}$ in direction of gradient of square loss

$\texttt{let}\ \mathbf{F} = [[i, s, p, c]]\ \texttt{in}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$

$\texttt{while(not converged)}\ \{$

$$\boldsymbol{\theta} = \lambda_{f_1 \in \mathbf{F}} \left(\boldsymbol{\theta}(f_1) - \frac{\alpha}{|\mathbf{Q}|} \underbrace{\sum_{x \in \mathtt{sup}(\mathbf{Q})} \mathbf{Q}(x) * \left(\sum_{f_2 \in \mathbf{F}} \boldsymbol{\theta}(f_2) * x[f_2] - x[u] \right) * x[f_1]}_{\text{Gradient of square loss}} \right)$$

$\}$

$\boldsymbol{\theta}$

(Simplified) Linear Regression in IFAQ

Batch Gradient Descent:
Update $\boldsymbol{\theta}$ in direction of gradient of square loss

$\texttt{let}$ $\mathbf{F} = [[i, s, p, c]]$ $\texttt{in}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$

$\texttt{while(not converged)}$ $\{$

$$\boldsymbol{\theta} = \lambda_{f_1 \in \mathbf{F}} \left(\boldsymbol{\theta}(f_1) - \frac{\alpha}{|\mathbf{Q}|} \underbrace{\sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * \left(\sum_{f_2 \in \mathbf{F}} \boldsymbol{\theta}(f_2) * x[f_2] - x[u] \right) * x[f_1]}_{\text{Gradient of square loss}} \right)$$

$\}$

$\boldsymbol{\theta}$

For simplicity and WLOG, we

1. set $\frac{\alpha}{|\mathbf{Q}|} = 1$
2. ignore $x[u]$

(Simplified) Linear Regression in IFAQ

$\texttt{let } \mathbf{F} = [[i, s, p, c]] \texttt{ in}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$

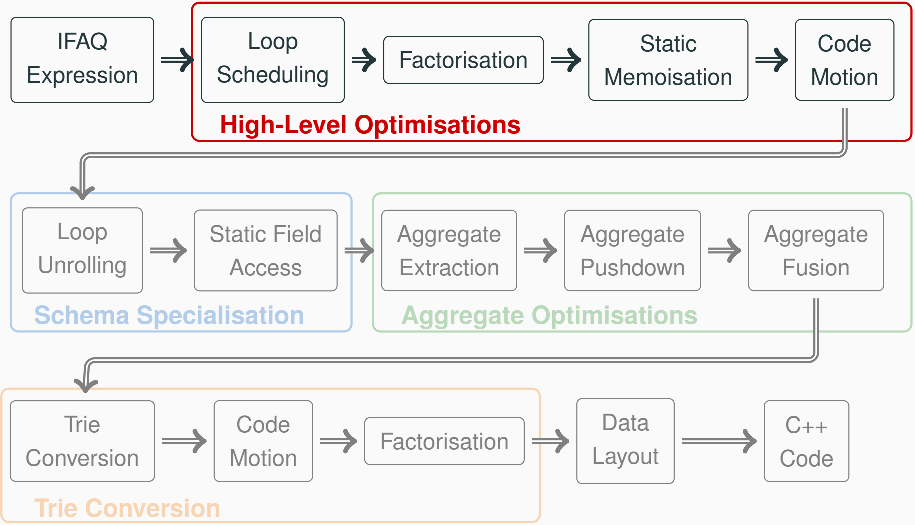
$\texttt{while(not converged) } \{$

$$\boldsymbol{\theta} = \lambda_{f_1 \in \mathbf{F}} \left(\boldsymbol{\theta}(f_1) - \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * \left(\sum_{f_2 \in \mathbf{F}} \boldsymbol{\theta}(f_2) * x[f_2] \right) * x[f_1] \right)$$

$\}$

$\boldsymbol{\theta}$

Next: High-Level Optimisations



High-Level Optimisations

Transformation Rule: **Normalisation**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) \; - \sum_{x \in \mathtt{sup}(\mathbf{Q})} \mathbf{Q}(x) \; * \; \sum_{f_2 \in \mathbf{F}} \Big(\; \theta(f_2) \; * \; x[f_2] \; \Big) \; * \; x[f_1] \right)$$

High-Level Optimisations

Transformation Rule: **Normalisation**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) - \sum_{x \in \mathtt{sup}(\mathbf{Q})} \mathbf{Q}(x) * \sum_{f_2 \in \mathbf{F}} \left(\theta(f_2) * x[f_2] \right) * x[f_1] \right)$$

Transformation Rule: **Normalisation**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) \;-\; \sum_{x \in \mathtt{sup}(\mathbf{Q})} \; \sum_{f_2 \in \mathbf{F}} \left(\mathbf{Q}(x) \;*\; \theta(f_2) \;*\; x[f_2] \;*\; x[f_1] \right) \right)$$

High-Level Optimisations

Transformation Rule: **Loop Scheduling**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) \; - \sum_{x \in \mathtt{sup}(\mathbf{Q})} \; \sum_{f_2 \in \mathbf{F}} \left(\mathbf{Q}(x) \; * \; \theta(f_2) \; * \; x[f_2] \; * \; x[f_1] \right) \right)$$

Order loops by size of support

High-Level Optimisations

Transformation Rule: **Loop Scheduling**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) \; - \sum_{f_2 \in \mathbf{F}} \; \sum_{x \in \mathtt{sup}(\mathbf{Q})} \left(\mathbf{Q}(x) \; * \; \theta(f_2) \; * \; x[f_2] \; * \; x[f_1] \right) \right)$$

Order loops by size of support

High-Level Optimisations

Transformation Rule: **Factorisation**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) \; - \sum_{f_2 \in \mathbf{F}} \; \sum_{x \in \mathtt{sup}(\mathbf{Q})} \Big(\mathbf{Q}(x) \; * \; \theta(f_2) \; * \; x[f_2] \; * \; x[f_1] \Big) \right)$$

High-Level Optimisations

Transformation Rule: **Factorisation**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) \ - \sum_{f_2 \in \mathbf{F}} \ \sum_{x \in \mathtt{sup}(\mathbf{Q})} \left(\mathbf{Q}(x) \ * \ \theta(f_2) \ * \ x[f_2] \ * \ x[f_1] \right) \right)$$

Less arithmetic operations

High-Level Optimisations

Transformation Rule: **Factorisation**

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) \ - \sum_{f_2 \in \mathbf{F}} \ \theta(f_2) \ * \sum_{x \in \mathtt{sup}(\mathbf{Q})} \left(\mathbf{Q}(x) \ * \ x[f_2] \ * \ x[f_1] \right) \right)$$

Less arithmetic operations

High-Level Optimisations

Transformation Rule: **Static Memoisation**

$$\theta \leftarrow \theta_0$$

$\texttt{while(not converged)}\{$

$$\theta = \lambda_{f_1 \in \mathbf{F}} \left(\theta(f_1) - \sum_{f_2 \in \mathbf{F}} \theta(f_2) * \sum_{x \in \texttt{sup}(\mathbf{Q})} \Big(\mathbf{Q}(x) * x[f_2] * x[f_1] \Big) \right)$$

$\}$

$$\theta$$

High-Level Optimisations

Transformation Rule: **Static Memoisation**

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$

$\texttt{while(not converged)}\{$

$$\boldsymbol{\theta} = \lambda_{f_1 \in \mathbf{F}} \left(\boldsymbol{\theta}(f_1) - \sum_{f_2 \in \mathbf{F}} \boldsymbol{\theta}(f_2) * \sum_{x \in \texttt{sup}(\mathbf{Q})} \left(\mathbf{Q}(x) * x[f_2] * x[f_1] \right) \right)$$

$\}$

$\boldsymbol{\theta}$

High-Level Optimisations

Transformation Rule: **Code Motion**

$$\theta \leftarrow \theta_0$$

$$\texttt{while(not converged)}\{$$

$$\texttt{let}\ \mathbf{M} = \lambda_{f_1 \in \mathbf{F}} \lambda_{f_2 \in \mathbf{F}} \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x[f_2] * x[f_1]\ \texttt{in}$$

$$\theta = \lambda_{f_1 \in \mathbf{F}} \Big(\theta(f_1) - \sum_{f_2 \in \mathbf{F}} \theta(f_2) * \mathbf{M}(f_1)(f_2) \Big)$$

$$\}$$

$$\theta$$

M defines the covariance matrix

Transformation Rule: **Code Motion**

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$

$\texttt{while(not converged)}\{$

$\quad \texttt{let } \mathbf{M} = \lambda_{f_1 \in \mathbf{F}} \lambda_{f_2 \in \mathbf{F}} \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x[f_2] * x[f_1] \texttt{ in}$

$\quad \boldsymbol{\theta} = \lambda_{f_1 \in \mathbf{F}} \Big(\boldsymbol{\theta}(f_1) - \sum_{f_2 \in \mathbf{F}} \boldsymbol{\theta}(f_2) * \mathbf{M}(f_1)(f_2) \Big)$

$\}$

$\boldsymbol{\theta}$

Transformation Rule: **Code Motion**

$$\texttt{let}\ \mathbf{M} = \lambda_{f_1 \in \mathbf{F}} \lambda_{f_2 \in \mathbf{F}} \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x[f_1] * x[f_2]\ \texttt{in}$$

$$\theta \leftarrow \theta_0$$

$$\texttt{while(not converged)}\{$$

$$\theta = \lambda_{f_1 \in \mathbf{F}} (\theta(f_1) - \sum_{f_2 \in \mathbf{F}} \theta(f_2) * \mathbf{M}(f_1)(f_2))$$

$$\}$$

$$\theta$$

High-Level Optimisations

Expression after High-Level Optimisations:

$$\texttt{let}\ \mathbf{M} = \lambda_{f_1 \in \mathbf{F}} \lambda_{f_2 \in \mathbf{F}} \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x[f_1] * x[f_2]\ \texttt{in}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$$

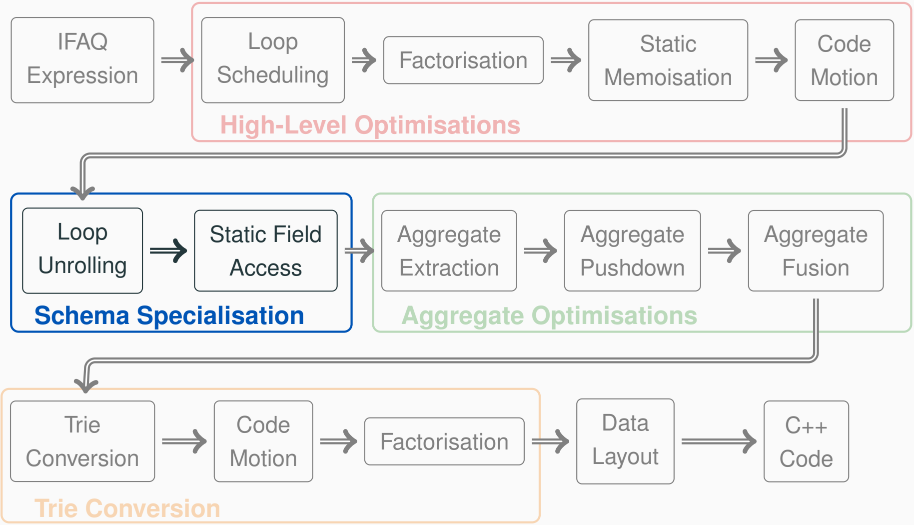
$$\texttt{while(not converged)}\{$$

$$\boldsymbol{\theta} = \lambda_{f_1 \in \mathbf{F}} (\boldsymbol{\theta}(f_1) - \sum_{f_2 \in \mathbf{F}} \boldsymbol{\theta}(f_2) * \mathbf{M}(f_1)(f_2))$$

$$\}$$

$$\boldsymbol{\theta}$$

Next: Schema Specialisation



Schema Specialisation

Transformation Rule: **Loop Unrolling**

$$\texttt{let}\ \mathbf{M} = \lambda_{f_1 \in \mathbf{F}} \lambda_{f_2 \in \mathbf{F}} \sum_{x \in \mathtt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x[f_1] * x[f_2]\ \texttt{in}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$$

$$\texttt{while(not converged)}\{$$

$$\boldsymbol{\theta} = \lambda_{f_1 \in \mathbf{F}} (\boldsymbol{\theta}(f_1) - \sum_{f_2 \in \mathbf{F}} \boldsymbol{\theta}(f_2) * \mathbf{M}(f_1)(f_2))$$

$$\}$$

$$\boldsymbol{\theta}$$

Unroll Loops over statically known features **F**

Schema Specialisation

Transformation Rule: **Loop Unrolling**

$$\texttt{let}\ \ \mathbf{M} = \lambda_{f_1 \in \mathbf{F}} \lambda_{f_2 \in \mathbf{F}} \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x[f_1] * x[f_2]\ \texttt{in}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$$

$$\texttt{while(not converged)}\{$$

$$\boldsymbol{\theta} = \left\{\left\{c \to \left(\boldsymbol{\theta}(c) - (\ldots + \boldsymbol{\theta}(c) * \mathbf{M}(c)(c) + \boldsymbol{\theta}(p) * \mathbf{M}(c)(p) \ldots)\right), \ldots\right\}\right\}$$

$$\}$$

$$\boldsymbol{\theta}$$

Unroll Loops over statically known features **F**

Schema Specialisation

Transformation Rule: **Loop Unrolling**

$$\texttt{let}\ \mathbf{M} = \left\{\left\{c \to \{\{\ldots, p \to \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x[c] * x[p], \ldots\}\}, \ldots\right\}\right\}\ \texttt{in}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$$

$$\texttt{while(not converged)}\{$$

$$\boldsymbol{\theta} = \left\{\left\{c \to \left(\boldsymbol{\theta}(c) - (\ldots + \boldsymbol{\theta}(c) * \mathbf{M}(c)(c) + \boldsymbol{\theta}(p) * \mathbf{M}(c)(p)\ldots)\right), \ldots\right\}\right\}$$

$$\}$$

$$\boldsymbol{\theta}$$

- Convert dictionaries over **F** into records
- Dynamic accesses into static accesses

Schema Specialisation

Transformation Rule: **Static Field Access**

$$\texttt{let}\ \ M = \left\{c = \left\{..., p = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.p, ..., \right\}, ...\right\}\ \texttt{in}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$$

$$\texttt{while(not converged)}\{$$

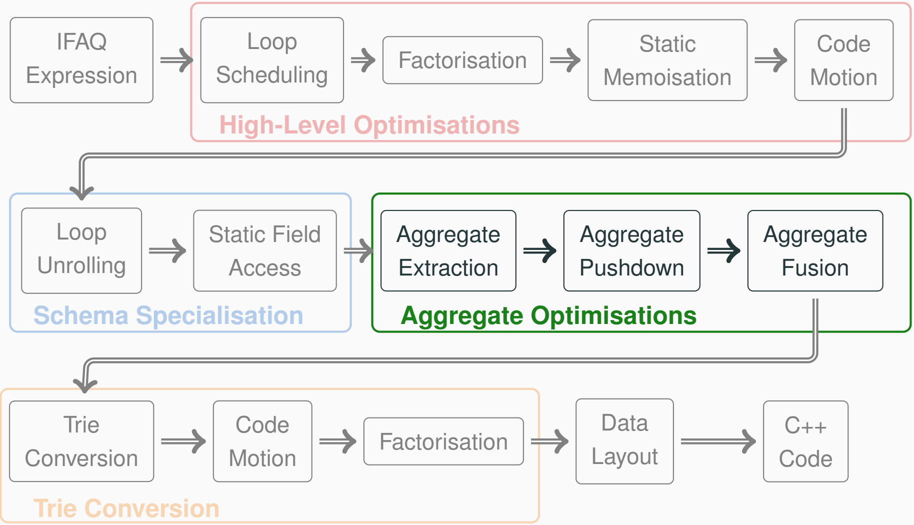
$$\theta = \left\{c = \theta.c - \left(... + \theta.c * M.c.c + \theta.p * M.c.p...\right), ...\right\}$$

$$\}$$

$$\boldsymbol{\theta}$$

- Convert dictionaries over **F** into records
- Dynamic accesses into static accesses

Next: Aggregate Optimisations



Transformation Rule: **Aggregate Extraction**

$$
\begin{aligned}
&\texttt{let } M = \\
&\quad \{c = \\
&\qquad \{\ldots, c = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.c, \;\; p = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.p, \ldots\} \\
&\quad , \ldots\} \texttt{ in } \ldots
\end{aligned}
$$

Aggregate Query Optimisations

Transformation Rule: **Aggregate Extraction**

$$\texttt{let}\ M =$$

$$\{c =$$

$$\{\ldots, c = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.c,\ \ p = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.p, \ldots\}$$

$$, \ldots\}\ \texttt{in}\ \ldots$$

Transformation Rule: **Aggregate Extraction**

$$\texttt{let}\ M_{cc} = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.c\ \texttt{in}$$

$$\texttt{let}\ M_{cp} = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.p\ \texttt{in}$$

$$\texttt{let}\ M = \{ \quad c = \{\ldots, \quad c = M_{cc}, \quad p = M_{cp}, \quad \ldots\}, \ldots\}\ \texttt{in}\ \ldots$$

Aggregate Query Optimisations

Recall: $\mathbf{Q}(x) = S(x_S) \bowtie I(x_I) \bowtie R(x_R)$

Transformation Rule: **Aggregate Extraction**

$$\texttt{let } M_{cc} = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.c \texttt{ in}$$

$$\texttt{let } M_{cp} = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.p \texttt{ in}$$

$$\texttt{let } M = \{ \quad c = \{..., \quad c = M_{cc}, \quad p = M_{cp}, \quad ...\}, ...\} \texttt{ in } ...$$

Aggregate Query Optimisations

Recall: $\mathbf{Q}(x) = S(x_S) \bowtie I(x_I) \bowtie R(x_R)$

Transformation Rule: **Aggregate Extraction**

$$\texttt{let}\ M_{cc} = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.c\ \texttt{in}$$

$$\texttt{let}\ M_{cp} = \sum_{x \in \texttt{sup}(\mathbf{Q})} \mathbf{Q}(x) * x.c * x.p\ \texttt{in}$$

$$\texttt{let}\ M = \{\quad c = \{\ldots,\quad c = M_{cc},\quad p = M_{cp},\quad \ldots\}, \ldots\}\ \texttt{in}\ \ldots$$

We can:

- avoid materialisation of $\mathbf{Q}(x)$
- inline code for join computation

Fast Join Recap

To compute $\mathbf{S}(x_S) \bowtie \mathbf{R}(x_R)$ on variable s:

1. Construct nested dictionaries over **R**:

$$\mathbf{H}_R = \sum_{x_R \in \texttt{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to \{\{x_R \to 1\}\}\}\}$$

For join value s, $\mathbf{H}_R(s)$ maps to partition of **R** with $s = x_R.s$

Fast Join Recap

To compute $\mathbf{S}(x_S) \bowtie \mathbf{R}(x_R)$ on variable s:

1. Construct nested dictionaries over **R**:

$$\mathbf{H}_R = \sum_{x_R \in \mathtt{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \rightarrow \{\{x_R \rightarrow 1\}\}\}\}$$

   For join value s, $\mathbf{H}_R(s)$ maps to partition of **R** with $s = x_R.s$

2. Iterate over **S**, and probe $\mathbf{H}_R$ for joining tuples:

$$\mathbf{J}_{S \bowtie R} = \sum_{x_S \in \mathtt{sup}(\mathbf{S})} \sum_{x_R \in \mathtt{sup}(\mathbf{H}_R(\{s = x_S.s\}))} \mathtt{let}\ k = \{s = x_S.s,\ i = x_S.i,\ u = x_S.u,\ c = x_R.c\}\ \mathtt{in}\ \{\{k \rightarrow \mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\})(x_R)\}\}$$

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \mathtt{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to \{\{x_R \to 1\}\}\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \mathtt{sup}(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to \{\{x_I \to 1\}\}\}\}$$

$$M_{cp} = \sum_{x_S \in \mathtt{sup}(\mathbf{S})} \sum_{x_R \in \mathtt{sup}(\mathbf{H}_R(\{s = x_S.s\}))} \sum_{x_I \in \mathtt{sup}(\mathbf{H}_I(\{i = x_S.i\}))}$$

$$\mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\})(x_R) * \mathbf{H}_I(\{i = x_S.i\})(x_I) * x_R.c * x_I.p$$

Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \mathrm{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to \{\{x_R \to 1\}\}\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \mathrm{sup}(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to \{\{x_I \to 1\}\}\}\}$$

$$M_{cp} = \sum_{x_S \in \mathrm{sup}(\mathbf{S})} \sum_{x_R \in \mathrm{sup}(\mathbf{H}_R(\{s = x_S.s\}))} \sum_{x_I \in \mathrm{sup}(\mathbf{H}_I(\{i = x_S.i\}))} \mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\})(x_R) * \mathbf{H}_I(\{i = x_S.i\})(x_I) * x_R.c * x_I.p$$

Push aggregate $\sum_{x_R} x_R.c$ into $\mathbf{H}_R$

Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \mathrm{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to x_R.c\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \mathrm{sup}(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to \{\{x_I \to 1\}\}\}\}$$

$$M_{cp} = \sum_{x_S \in \mathrm{sup}(\mathbf{S})} \cancel{\sum_{x_R \in \mathrm{sup}(\mathbf{H}_R(\{s = x_S.s\}))}} \sum_{x_I \in \mathrm{sup}(\mathbf{H}_I(\{i = x_S.i\}))}$$

$$\mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\})\cancel{(x_R)} * \mathbf{H}_I(\{i = x_S.i\})(x_I) * x_I.p$$

Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \mathtt{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to x_R.c\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \mathtt{sup}(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to \{\{x_I \to 1\}\}\}\}$$

$$M_{cp} = \sum_{x_S \in \mathtt{sup}(\mathbf{S})} \sum_{x_I \in \mathtt{sup}(\mathbf{H}_I(\{i = x_S.i\}))}$$

$$\mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\}) * \mathbf{H}_I(\{i = x_S.i\})(x_I) * x_I.p$$

Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \mathrm{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to x_R.c\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \mathrm{sup}(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to \{\{x_I \to 1\}\}\}\}$$

$$M_{cp} = \sum_{x_S \in \mathrm{sup}(\mathbf{S})} \sum_{x_I \in \mathrm{sup}(\mathbf{H}_I(\{i = x_S.i\}))} \mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\}) * \mathbf{H}_I(\{i = x_S.i\})(x_I) * x_I.p$$

Push aggregate $\sum_{x_I} x_I.p$ into $\mathbf{H}_I$

Aggregate Query Optimisations

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \sup(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \rightarrow x_R.c\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \sup(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \rightarrow x_I.p\}\}$$

$$M_{cp} = \sum_{x_S \in \sup(\mathbf{S})} \cancel{\sum_{x_I \in \sup(\mathbf{H}_I(\{i = x_S.i\}))}}$$

$$\mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\}) * \mathbf{H}_I(\{i = x_S.i\})(\cancel{x_I})$$

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \sup(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to x_R.c\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \sup(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to x_I.p\}\}$$

$$M_{cp} = \sum_{x_S \in \sup(\mathbf{S})} \mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\}) * \mathbf{H}_I(\{i = x_S.i\})$$

Transformation Rule: **Aggregate Pushdown**

$$\mathbf{H}_R = \sum_{x_R \in \text{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to x_R.c\}\}$$

$$\mathbf{H}_I = \sum_{x_I \in \text{sup}(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to x_I.p\}\}$$

$$M_{cp} = \sum_{x_S \in \text{sup}(\mathbf{S})} \mathbf{S}(x_S) * \mathbf{H}_R(\{s = x_S.s\}) * \mathbf{H}_I(\{i = x_S.i\})$$

$$\mathbf{H}'_R = \sum_{x_R \in \text{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to x_R.c * x_R.c\}\}$$

$$\mathbf{H}'_I = \sum_{x_I \in \text{sup}(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to 1\}\}$$

$$M_{cc} = \sum_{x_S \in \text{sup}(\mathbf{S})} \mathbf{S}(x_S) * \mathbf{H'}_R(\{s = x_S.s\}) * \mathbf{H'}_I(\{i = x_S.i\})$$

Similarly for M_{cc}

Aggregate Query Optimisations

Transformation Rule: **Aggregate Fusion**

$$\mathbf{H}_R = \sum_{x_R \in \sup(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \rightarrow x_R.c\}\}$$

$$\mathbf{H}'_R = \sum_{x_R \in \sup(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \rightarrow x_R.c * x_R.c\}\}$$

Fuse $\mathbf{H}_R$ and $\mathbf{H}'_R$

Transformation Rule: **Aggregate Fusion**

$$\mathbf{H}''_R = \sum_{x_R \in \mathrm{sup}(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to \{v_R = x_R.c, \; v'_R = x_R.c * x_R.c\}\}\}$$

$\mathbf{H}''_R$ computes two aggregates

Aggregate Query Optimisations

Transformation Rule: **Aggregate Fusion**

$$\mathbf{H}''_R = \sum_{x_R \in \sup(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \rightarrow \{v_R = x_R.c,\ \ v'_R = x_R.c * x_R.c\}\}\}$$

$$\mathbf{H}''_I = \sum_{x_I \in \sup(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \rightarrow \{v_I = x_I.p,\ \ v'_I = 1\}\}\}$$

Fuse $\mathbf{H}_I$ and $\mathbf{H}'_I$

Aggregate Query Optimisations

Transformation Rule: **Aggregate Fusion**

$$\mathbf{H}''_R = \sum_{x_R \in \sup(\mathbf{R})} \mathbf{R}(x_R) * \{\{\{s = x_R.s\} \to \{v_R = x_R.c,\ \ v'_R = x_R.c * x_R.c\}\}\}$$

$$\mathbf{H}''_I = \sum_{x_I \in \sup(\mathbf{I})} \mathbf{I}(x_I) * \{\{\{i = x_I.i\} \to \{v_I = x_I.p,\ \ v'_I = 1\}\}\}$$

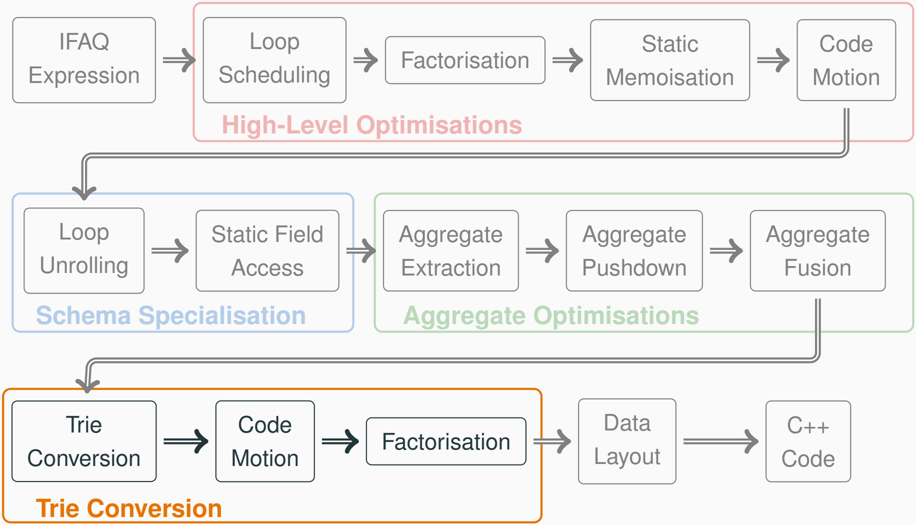
$$M_{cc,cp} = \sum_{x_S \in \sup(\mathbf{S})} \mathbf{S}(x_S) * \Big($$

$$\texttt{let}\ \ w_R = H''_R(\{s = x_S.s\})\ \texttt{in}$$

$$\texttt{let}\ \ w_I = H''_I(\{i = x_S.i\})\ \texttt{in}$$

$$\{m_{cp} = w_R.v_R * w_I.v_I,\quad m_{cc} = w_R.v'_R * w_I.v'_I\}\Big)$$

Next: Trie Conversion



Transformation Rule: **Trie Conversion**

$$\boldsymbol{H}''_R = \sum_{x_r \in \mathtt{sup}(\mathbf{R})} \mathbf{R}(x_r) * \{\{\{s = x_r.s\} \rightarrow \{v_R = x_r.c,\ v'_R = x_r.c * x_r.c\}\}\}$$

$$\boldsymbol{H}''_I = \sum_{x_i \in \mathtt{sup}(\mathbf{I})} \mathbf{I}(x_i) * \{\{\{i = x_i.i\} \rightarrow \{v_I = x_i.p,\ v'_R = 1\}\}\}$$

$$M_{cc,cp} = \sum_{x_S \in \mathtt{sup}(\mathbf{S})} \mathbf{S}(x_S) * \Big(\mathtt{let}\ \ w_R = \mathbf{H}''_R(\{s = x_S.s\})\ \mathtt{in}\ \ \mathtt{let}\ \ w_I = \mathbf{H}''_I(\{i = x_S.i\})\ \mathtt{in}\ \ \{m_{cp} = w_R.v_R * w_I.v_I,\quad m_{cc} = w_R.v'_R * w_I.v'_I\} \Big)$$

Turn relations into tries (i.e., nested dictionaries)

Trie Conversion

Transformation Rule: **Trie Conversion**

$$M_{cc,cp} = \sum_{x_S \in \texttt{sup}(\mathbf{S})} \mathbf{S}(x_S) * \Big($$

$$\texttt{let } w_R = \mathbf{H}''_R(\{s = x_S.s\}) \texttt{ in}$$

$$\texttt{let } w_I = \mathbf{H}''_I(\{i = x_S.i\}) \texttt{ in}$$

$$\{m_{cp} = w_R.v_R * w_I.v_I, \quad m_{cc} = w_R.v'_R * w_I.v'_I\}\Big)$$

Turn relation S into trie S’

Transformation Rule: **Trie Conversion**

One loop for each join variable

$$M_{cc,cp} = \sum_{x_S \in \texttt{sup}(\mathbf{S'})} \sum_{x_i \in \texttt{sup}(\mathbf{S'}(x_s))} S'(x_s)(x_i) * \Big(\texttt{let } w_R = \mathbf{H}''_R(\{s = x_S.s\}) \texttt{ in } \texttt{let } w_I = \mathbf{H}''_I(\{i = x_S.i\}) \texttt{ in } \{m_{cp} = w_R.v_R * w_I.v_I, \quad m_{cc} = w_R.v'_R * w_I.v'_I\} \Big)$$

Trie Conversion

Transformation Rule: **Code Motion**

Move up the look-up into $\mathbf{H}_R$

$$M_{cc,cp} = \sum_{x_S \in \texttt{sup}(\mathbf{S'})} \sum_{x_i \in \texttt{sup}(\mathbf{S'}(x_s))} S'(x_s)(x_i) * \Big(\texttt{let}\ \ w_R = \mathbf{H}''_R(\{s = x_S.s\})\ \texttt{in}\ \ \texttt{let}\ \ w_I = \mathbf{H}''_I(\{i = x_S.i\})\ \texttt{in}\ \ \{m_{cp} = w_R.v_R * w_I.v_I,\quad m_{cc} = w_R.v'_R * w_I.v'_I\} \Big)$$

Trie Conversion

Transformation Rule: **Code Motion**

Move up the look-up into $\mathbf{H}_R$

$$M_{cc,cp} = \sum_{x_S \in \texttt{sup}(\mathbf{S'})}$$

$$\texttt{let } w_R = \mathbf{H}''_R(\{s = x_S.s\}) \texttt{ in}$$

$$\sum_{x_i \in \texttt{sup}(\mathbf{S'}(x_s))} S'(x_s)(x_i) * \Big($$

$$\texttt{let } w_I = \mathbf{H}''_I(\{i = x_S.i\}) \texttt{ in}$$

$$\{m_{cp} = w_R.v_R * w_I.v_I, \quad m_{cc} = w_R.v'_R * w_I.v'_I\}\Big)$$

Transformation Rule: **Factorisation**

Less arithmetic operations

$$M_{cc,cp} = \sum_{x_S \in \texttt{sup}(\mathbf{S'})}$$

$$\texttt{let } w_R = \mathbf{H}''_R(\{s = x_S.s\}) \texttt{ in}$$

$$\sum_{x_i \in \texttt{sup}(\mathbf{S'}(x_s))} S'(x_s)(x_i) * \Big($$

$$\texttt{let } w_I = \mathbf{H}''_I(\{i = x_S.i\}) \texttt{ in}$$

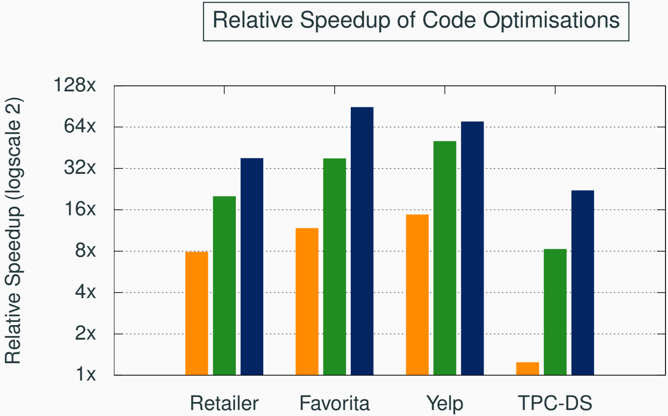
$$\{m_{cp} = w_R.v_R * w_I.v_I, \quad m_{cc} = w_R.v'_R * w_I.v'_I\}\Big)$$

Trie Conversion

Transformation Rule: **Factorisation**

Less arithmetic operations

$$M_{cc,cp} = \sum_{x_S \in \texttt{sup}(\mathbf{S'})} \texttt{let } w_R = \mathbf{H}''_R(\{s = x_S.s\}) \texttt{ in } \{m_{cp} = w_R.v_R, \quad m_{cc} = w_R.v'_R\} * \sum_{x_i \in \texttt{sup}(\mathbf{S'}(x_s))} S'(x_s)(x_i) * \Big(\texttt{let } w_I = \mathbf{H}''_I(\{i = x_S.i\}) \texttt{ in } \{m_{cp} = w_I.v_I, \quad m_{cc} = w_I.v'_I\}\Big)$$



Added optimisations for covariance matrix computation:

specialisation → + sharing → + parallelisation

AWS d2.xlarge (4 vCPUs, 32GB)

Conclusion

Three-step recipe for efficient machine learning over databases:

1. Turn the learning problem into a database problem

2. Exploit the problem structure to lower the complexity

3. Specialise and optimise the code to lower the constant factors

`Q.E.D.`